COMPARISON OF DEEP NETWORKS FOR GESTURE RECOGNITION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BUĞRA SOFU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

COMPARISON OF DEEP NETWORKS FOR GESTURE RECOGNITION

submitted by **BUĞRA SOFU** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. İlkay Ulusoy
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Prof. Dr. Alptekin Temizel
Graduate School of Informatics, METU _____

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Assist. Prof. Dr. Hamdi Dibeklioğlu
Computer Engineering, Bilkent University _____

Date: 06.09.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Buğra Sofu

Signature :

ABSTRACT

COMPARISON OF DEEP NETWORKS FOR GESTURE RECOGNITION

Sofu, Buğra
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. İlkay Ulusoy

September 2021, 105 pages

Gesture recognition is an important problem and has been studied over the years especially in the fields such as surveillance systems, analysis of human behavior, robotics etc. In this thesis, different state of art algorithms, which are based on deep learning, were implemented and compared considering model complexities and accuracies. Also, a new approach was proposed and compared with them. Tested algorithms can be classified into two main categories: hybrid approaches, which use CNN and LSTM architectures successively, and three dimensional convolutional neural networks (3D-CNNs). For the hybrid approaches, we studied CNN-LSTM models and investigated the effect of different feature extractors such as Inception-V3 and ResNext50 models. For the ResNext50 architecture, additional to original network, we included an attention model called Squeeze and Excitation Block (SE). By this new approach, 21% accuracy increase was reached while the number of parameters was decreased, which means less model complexity than the original approach. For the 3D-CNNs, I3D model, which has pre-trained ImageNet weights, was applied and compared with C3D models, which cannot use ImageNet weights directly. Ability to use ImageNet weights gives the advantage of fast training, since network is initialized with ImageNet features, and can also result in a more accurate and effective model overall.

16.5% accuracy increase was obtained for the 3D-CNN architecture when I3D model was trained on Kinetics dataset.

Keywords: Gesture Recognition, Hybrid Networks, 3D-CNNs, Two Stream Networks

ÖZ

İFADE TANIMA PROBLEMİ İÇİN FARKLI DERİN AĞLARIN KARŞILAŞTIRMASI

Sofu, Buğra
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İlkay Ulusoy

Eylül 2021 , 105 sayfa

İfade tanıma problem, güvenlik sistemleri, insan davranış analizi, robotik gibi alanlarda bilgisayar bilimleri tarafından uzun yıllardır çalışılan bir problemdir. Bu tezde, derin öğrenme tabanlı, literatürede bulunan güncel algoritmalar model karmaşıklığına karşılık başarım oranı gözetilerek birbiri ile karşılaştırmıştır. Ayrıca, güncel modelleri kullanarak alternatif bir model önerilmiştir.Test edilen yöntemler; CNN ve LSTM yapılarını ardışık olarak kullanan hibrit yapılar ve üç boyutlu evrişimsel sinir ağları olmak üzere iki ana başlıkta incelenmiştir. Hibrit yöntemler için CNN-LSTM ağ yapıları kullanılmıştır. Bu ağ yapılarını incelerken asıl amaç, farklı öz nitelik çıkartma ağlarının toplam performansa etkisinin gözlemlenmesiydi. Inception-V3 ve ResNext50 ağ modelleri öz nitelik tanımlayıcı olarak kullanılmıştır. ResNext50 modelinde ayrıca “Squeeze and Excitation” (SE) blok yapısı da kullanılmıştır. CNN-LSTM modellerinde kullanılan öznitelik çıkarma modellerini değiştirerek 21% oranında başarım artışı gözlemlenmiştir. İkinci olarak, üç boyutlu evrişimsel sinir ağları üzerinde çalışılmıştır (3D-CNN). Burada, ImageNet parametrelerini kullanmaya imkan tanıyan, I3D moleli üzerinde testler yapılmıştır. Bu testlerde, ImageNet parametrelerini

doğrudan kullanmayan C3D modelleriyle karşılaştırarak, büyük verisetlerinde eğitilmiş parametrelerin kullanımının model başarımına etkisini gözlemlenmiştir. Testler sonucunda, Kinetics veri setinde eğitilmiş I3D modelini kullanarak 16.5% oranında başarım artışı gözlemlenmiştir. Elde edilen sonuçları güncel yayınları karşılaştırma imkanı tanıdığı ve literatürdeki modellerden yararlanarak alternatif bir çözüm önerisi sunduğu için faydalı olacağını değerlendiriyoruz.

Anahtar Kelimeler: İfade Tanıma, Hibrit Yapılar, 3D-CNNs, İkili Akış Ağları

To my dearest family.

ACKNOWLEDGMENTS

Before all else, I would like to express my admiration to my supervisor Prof. Dr. İlkay Ulusoy for her guidance and positive attitude. She was really open-minded and supportive throughout my study.

Also I wish to thank my colleagues; Arda Özertem, Onur Akın and Serdar Çakır and my friend Mustafa Erseven for reviewing my thesis and their valuable feedbacks.

Last but not least, I would like to thanks my parents Nevin and Doğan Sofu and my dear sister Büşra Sofu for their support, encouragement and confidence throughout the years of my education.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
ResNet	Residual Network
RGB	Red Green Blue
SVM	Support Vector Machine
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
ConvNet	Convolutional Network
RELU	Rectified Linear Units
FC	Fully-Connected
SGD	Stochastic Gradient Descent
I3D	Inflated 3 Dimensional
ICPR	International Association forPattern Recognition

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Gesture based communication is part of our lives. We, human beings, strongly use our hands and mimics while we are communicating each other. Deaf community entirely uses this kind of communication; sign languages. Additional to the sign language, gesture based communication is used a lot, especially to inform others that are not near to speak directly. For example a police officer could use gesture based communication to guide the traffic, or a referee to state some decisions about the game. Humans, by their nature, are capable of learning to interpret these kind of patterns in daily life.

When we examine the gesture recognition problem, we can say that; compared to the static images, it is hard to develop an algorithm to understand the action for the given video input, since videos contain more information. Gesture could be performed in different environments, different background and different angles. Our model should be general enough to recognize these kind of scenarios.

1.2 Applied Methods and Models

When we look at the literature, different approaches have been proposed. Since gesture recognition problem can be seen as recognition of time series video frames, any sequential learning algorithm for time series inputs can be used to model the system [22], [23], [24], [25], [26].

To train the model, features should be extracted to describe the given model data. For the last few years rather than using hand crafted features ([27], [28]), feature learning based methods have become popular. In deep learning, instead of constructing hand-crafted features, networks learn the features, by that way feature construction process becomes automatic. Deep networks could be used for the feature learning process and sequential learning algorithms could be used for the training part. By that way we can create a hybrid network to take advantage of their strengths.

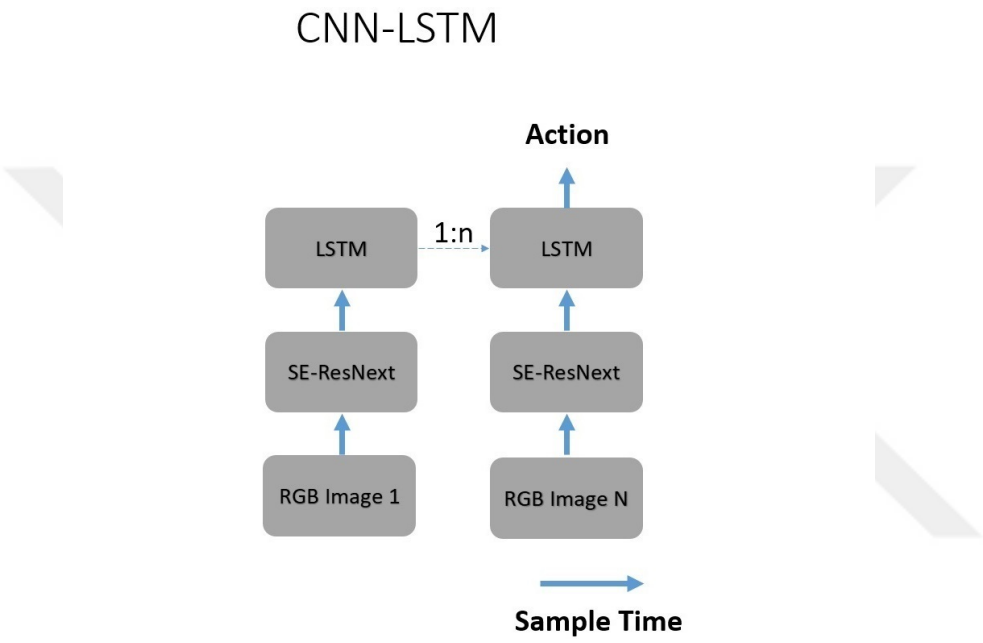


Figure 1.1: Different architectures in the literature.

One alternative for sequential learning algorithm is to use HMMs to model the system. HMM has been widely used for speech recognition tasks for the last centuries, but any time series based model can be trained using HMMs ([29], [30]). Another popular model is recurrent neural networks[23], [31]. Instead of HMMs we could use RNN to learn the sequential model as well. Again we could use CNNs as a feature extractor and RNNs for sequential learning. RNN is a sequential learning algorithm that can memorize/remember previous inputs in memory. RNNs can be seen as a sequence of traditional neural networks, every network takes the input that comes before it, so they are called recurrent. RNNs apply same operation for all of the given sequence. Since each calculated outputs are fed to next inputs, inputs and outputs depend on

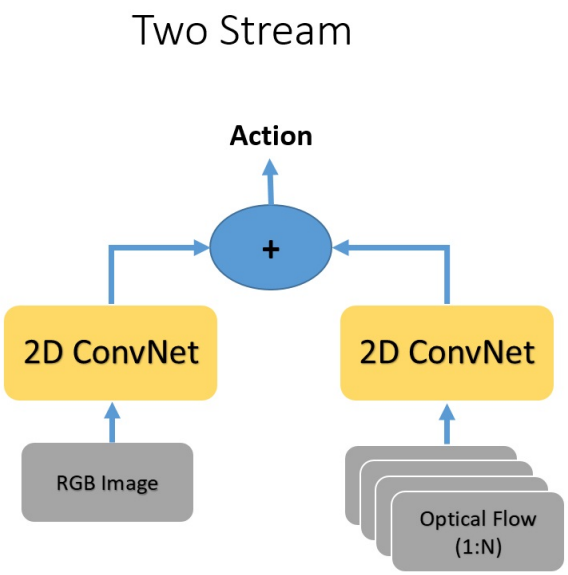


Figure 1.2: Different architectures in the literature.

each other. For the traditional neural networks, this is not the case (inputs and outputs are independent of each other).

Second alternative is to construct 3D-CNN to learn both special and temporal features with one model ([32], [20]). In 3D-CNN, the kernel slides in 3 dimensions. We could use3D-CNNs with video inputs that are also stack of images. Input shape for a 3D-CNN would have four dimensions (N, W, H, C). Here, N is the number of images that would be fed to the network, W and H are the width and height of the images and C is the number of channel. Similarly, 2D images would have 3 dimensions (W, H, C). For 3D-CNNs again we have fully connection for the colour channels and local connections for the rest. Recently 3D CNNs are created by inflating the 2D architectures such as Inception [21], which could benefit from pre-trained ImageNet weights.

For the recent publications, people try to learn features from different image sources such as optical flow images, then try to merge the results. Such networks are called "Two-Stream Convolutional Networks [1]". Input videos have both spatial and temporal information, that are also called spatiotemporal. So, we need to learn both spatial and temporal features. The idea behind the two stream networks is to create

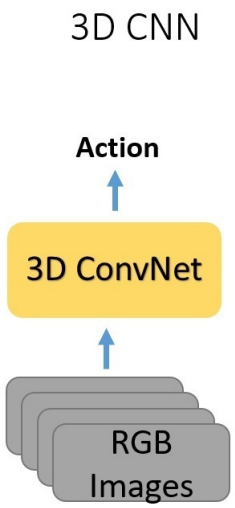


Figure 1.3: Different architectures in the literature.

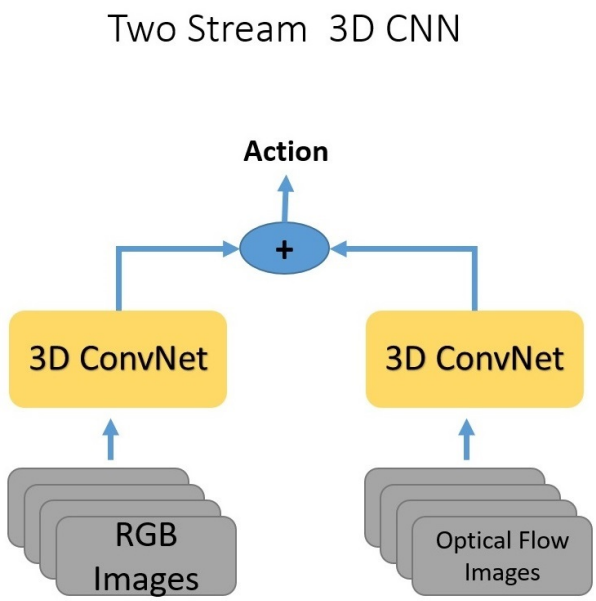


Figure 1.4: Different architectures in the literature.

two networks, one for spatial features and another for temporal features. In this idea, spatial network computes information about the different objects in the scene and

temporal stream network computes information about the displacement of objects in the stream.

Lastly we would like to mention the different ideas to improve the CNN architectures. One interesting idea is proposed by [16], "Squeeze and Excitation Networks" (SE). Squeeze and Excitation block can be added to CNNs and it can increase the feature description capability of the network. Main idea of SE block dynamically excites feature maps that help classification and suppress feature maps that don't help based on the patterns of global averages of feature maps. SE block improves channel interdependencies at almost no computational cost and it can be added existing CNN architectures. It was used "ImageNet competition" in 2017.

Different proposed architectures are illustrated in Figure 1.1 to Figure 1.4. In 1.1 CNN-LSTM hybrid, in 1.2 Two Stream scenario are depicted. In 1.3, traditional 3D-CNN is demonstrated. Finally in 1.4 Two Stream version of 3D-CNN is given.

1.3 Contributions and Novelties

In this thesis, we have implemented different methods that are introduced in the previous section. We have trained various CNN-LSTM hybrid shown in Figure 1.1. Different feature extractor networks were used such as InceptionV3, InceptionResNetV2, ResNext, MobileNet and Xception.

We have also combined two different design model. Proposed architecture is given in Figure 1.5. We have used CNN-LSTM networks with Two Stream configuration, which can be seen as combination of Figure 1.1 and Figure 1.2. With this configuration, we have obtained promising results. For the CNN architecture we have used ResNext architecture with Squeeze and Excitation Network model, which is one of the best feature descriptor. We have trained two networks, one stream is trained with RGB images and the other is trained with flow images. Final result is obtained by averaging the outputs. We computed optical flow with a TV-L1 algorithm [33].

Similar to our model, Zhao et al [34] used two stream architecture with CNN-LSTM hybrid. Here, they used CNN-LSTM hybrid only for RGB inputs. For the temporal

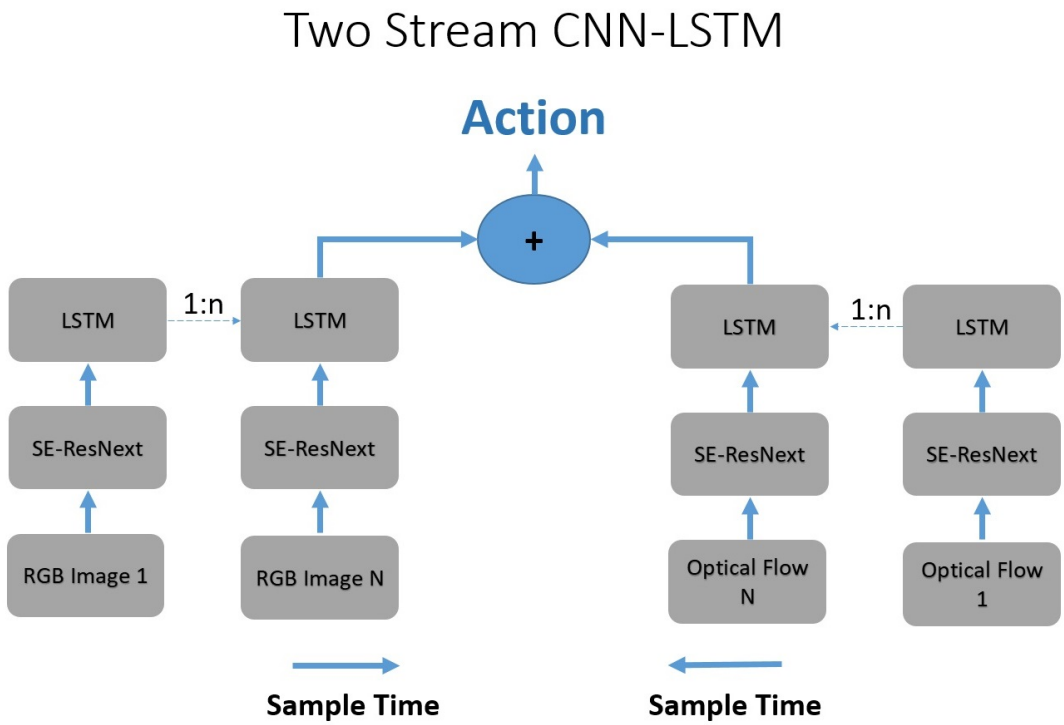


Figure 1.5: Proposed Method.

network, they trained DenseNet with optical flow inputs. Detailed architecture and implementation details will be introduced in Chapter 3.

For the 3D-CNN models, we have created a general model, known as C3D model, based on Figure 1.3. Finally we have trained I3D model, given in Figure 1.4.

By creating different networks, we had the chance of comparing different models. Throughout the thesis, results were analysed over two main categories. These are CNN-LSTM hybrids and 3D-CNN models. We have investigated when to use which architecture and pros and cons of these different methods.

1.4 The Outline of the Thesis

In Chapter 1; problem definition, proposed solutions so far and our solution architecture is given. Chapter 2 will introduce the models and how to combine them in details. Chapter 3 will give the experimental setup, data set and obtained results. Here, obtained results will be discussed in detail. Chapter 4 is the conclusion and

here obtained results will be summarized. Finally in Chapter 5 possible future works will be discussed and additional contributions will be addressed.

CHAPTER 2

THEORETICAL BACKGROUND AND RELATED WORKS

2.1 Optical Flow

This section will introduce optical flow method. We will be talking about general optical flow formulation and the method we have used for our algorithms [8].

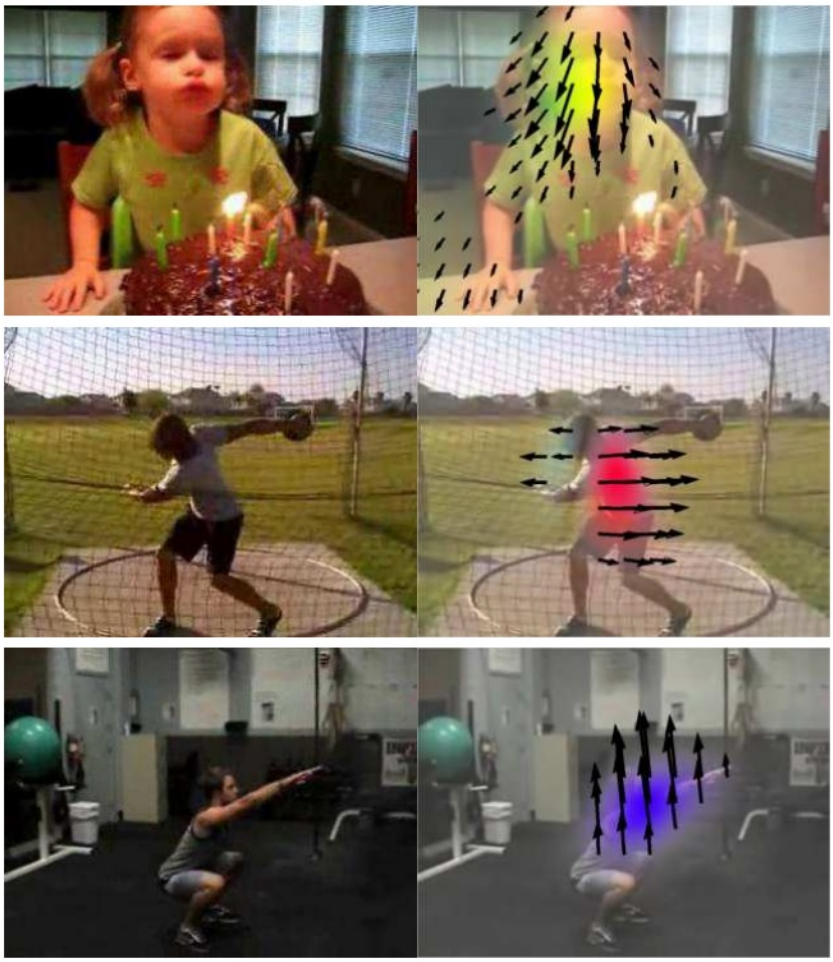
For action classification task, successive frames carry valuable information. To classify the action correctly, we need not only learn spatial features, but also we need temporal features. The best way to capture this temporal information is to capture the difference in displacement of optical flow between the video frames.

Optical flow technics try to estimate dense motion filed that are corresponding to the displacement of each pixel. Naturally, there are challenges in this area such as motion discontinuities, large displacements, illumination changes or computational complexities.

Optical flow algorithms generally calculate displacement of object on 2D image plane, although in practice we would like to learn the real displacement of the object in the 3D world. Calculating the displacement in real world is known as motion field, but motion field and optical flow the terms are mixed up mostly.

Since the optical flow calculation is done in image plane, actually we are trying to represent motion of intensities in the image plane. Optical flow calculation assumes that intensity of moving pixels remains constant during motion. But in real life, this is not the case and cause a problem known as aperture problem [8].

The visualization of motion fields provides a better understaing. There are two main



(a) Input Image (b) Prediction

Figure 2.1: Arrow and colour code visualization methods. Sample picture is taken from [7].

way to visualize optical flow; arrow visualization or colour code visualization. Arrow visualization gives good information for the direction of the objects. On the other hand colour code visualization gives a clearer visualization without overlapping arrows.

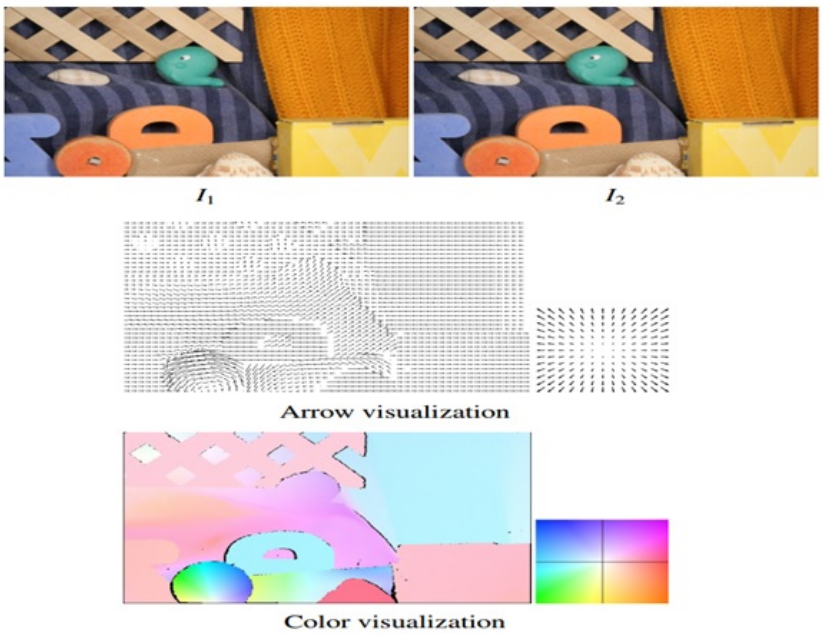


Figure 2.2: Comparison of different optical flow visualization methods. Sample picture is taken from [8].

2.1.1 Estimation Principles

In order to calculate optical flow between successive images, we need an assumption to correlate pixels over time. The most general assumption is "intensity remains constant during displacement", which is called the brightness constancy constraint. Brightness constancy constraint equation (BCCE) is then defined in continuous space as ([35], [36], [33]):

$$\frac{dI}{dt}(x(t), t) = 0 \tag{2.1}$$

Where I represents intensity, t represents time and x(t) is the given intensity map over

time. This can be re-write in discrete space as;

$$
\begin{aligned}
&I(x + w(x), t + 1) - I(x, t) = 0 \\
&w(x) = (u(x), v(x))^T \\
&x = (x_1, x_2)^T
\end{aligned}
\tag{2.2}
$$

Here w represents target displacement vector. To further simplify the solution, we can represent equation 2.2 with partial derivatives as;

$$
\frac{\delta I}{\delta x_1}(x)u(x) + \frac{\delta I}{\delta x_2}(x)v(x) + \frac{\delta I}{\delta t}(x) = 0 \tag{2.3}
$$

With brightness constancy constraint assumption, we got one equation with two unknown. Without any additional information from neighbouring context or additional constrain that encoding a priori information, 2D problem remains under constrained, which is called as aperture problem. This a priori could be given as either local or global constraints.

For local constrains, different approaches are derived that impose the flow field to follow a parametric model. Motion based parametric models or neighbourhood selection based approaches are used. On the other hand for the global constrains, regularization models could be used. These models mostly adopt smooth definition of motion field.

2.1.2 A Duality Based Approach for Realtime TV-L1 Optical Flow

In practice, brightness constancy assumption is impractical due to imperfect photometric expression of the real physical motion in the scene. One example could be given as moving light source in a static scene. In this case there would be brightness variation while there is no motion in the scene. Although it is possible to create synthetic scene which holds the constraint, in practice it is not possible due to change of illumination source of the scene, shadows and specular reflections.

Zach et al proposed regularization based method based on total variation regularization and L1 data penalty ([35], [36], [33]). The deviation from the brightness

constancy constraint is penalized with a function that is defined as;

$$\rho_{data} = (x, I_1, I_2, w) = \phi(I_2(x + w(x)) - I_1(x)) \tag{2.4}$$

Here, I_1 = I(.,t) and I_2 = I(.,t+1) denote two successive frames and ϕ is the penalty function. Using this penalty with a regularization term, we can minimize target disparity map w (u(x),v(x)) as;

$$\int_\Omega \{\lambda\phi(I_0(x) - I_1(x + w(x))) + \psi(w, \nabla w, ...)\}dx \tag{2.5}$$

Where ϕ is the penalty function and $\psi(w, \nabla w)$ is the regularization term. λ weights between the penalty function and regularization term. Selecting ϕ(x) = |x| and $\psi(\nabla w)$ = | ∇w | yields the L1 data penalty term and total variation regularization;

$$E = \int_\Omega \{\lambda|(I_0(x) - I_1(x + w(x))| + |\nabla w|\}dx \tag{2.6}$$

To solve equation 2.6 first we linearize I_1 near x+w_0 using first order Taylor approximation;

$$I_1(x + w) = I_1(x + w_0) + < \nabla I_1, w - w_0 > \tag{2.7}$$

Where w_0 is a given disparity map and ∇I_1 is the partial derivatives of the image intensity w.r.t. w.

Putting this linearization into the equation 2.6 we get;

$$E = \int_\Omega \{\lambda|w\nabla I_1 + I_1(x + w_0) - w_0\nabla I_1 - I_0| + |\nabla w|\}dx \tag{2.8}$$

We minimize equation 2.8 to solve the optical flow equation.

2.2 Two Stream Networks

Video inputs contain a lot more information, compared to still images. Temporal information gives us important clues about motion information to recognize the given action. For action classification, challenge is to capture both motion based features and spatial features at the same time.

Before deep learning methods become popular, most of the algorithms for video classification is based on shallow high-dimensional encodings of local spatio-temporal features. People generally try to extract features like; Histogram of Oriented Gradients (HoG) or Histogram of Optical Flow (HoF), then try to encode these features with Bag of Features (BoF) and combine with an SVM classifier.

To understand an action, human visual system fuses object recognition (ventral stream) and motion recognition (dorsal stream) based information. Simonyan et al [1] proposed a new architecture that works similar manner, which is called "Two Stream Networks". They have used two separate recognition networks; one network to learn spatial features and another for temporal features. This idea is illustrated in Figure 2.3.

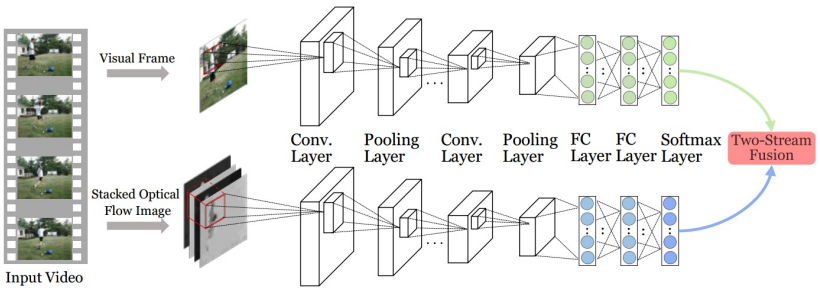


Figure 2.3: Proposed Two StreamNet architecture in [9].

Videos have both spatial and temporal information, which are also called spatiotemporal. So, we need to learn both spatial and temporal features. The idea behind the two stream networks is to create two networks, one for spatial features and another for temporal features, such that;

- Spatial network: Computes information about the scenes and different objects

present in the video. Static frames still carries lots of good features to describe the scene. There are well known networks to extract these features such as VGG16, Inception etc.

- Temporal stream network: Computes information about temporal features. The best way to capture this temporal information is to capture the difference in displacement of optical flow from the video frames. Stacked optical flow images is fed to this networks, by that way we don't need to estimate motion implicitly, since optical flow images naturally carry this information.

2.2.1 Temporal stream network configuration

There are two popular alternative to calculating temporal information, which are optical flow stacking and trajectory stacking. Dense optical flow images can be seen as a set of displacement vector fields between successive frames. For the given pixel location d(u,v) denote the displacement vector at that point. As describe in the previous chapter, for convenience of the calculation, displacement can be calculated in discrete domain with partial derivatives; dxt and dyt denote the horizontal and vertical component of the vector field. By calculating the displacement vectors for every pixel, two channel images are created that will be fed to the temporal network. An alternative to optical flow, trajectory based representation could also be used. Here, instead of sampling at same location across several frames using flow information, samples are taken along motion trajectories.

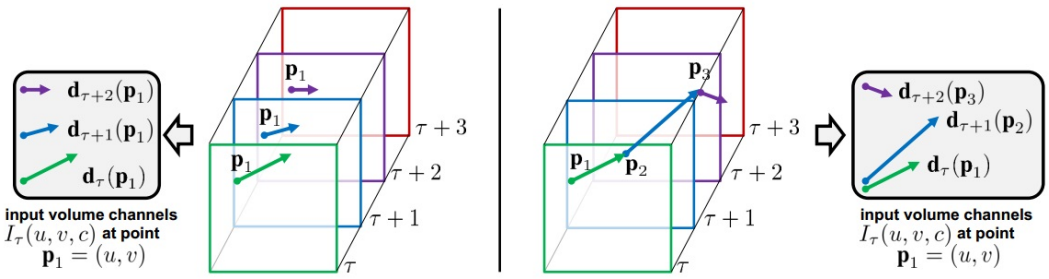


Figure 2.4: Optical Flow Stacking vs Trajectory Stacking discussed in [1].

Temporal stream images will be constructed for optical flow stacking as ([1]);

$$\begin{aligned} I_T(u,v,1) &= d^x_{T+k-1}(u,v) \\ I_T(u,v,2) &= d^y_{T+k-1}(u,v) \\ u = [1:w], v &= [1,h], k = [1:L] \end{aligned} \tag{2.9}$$

And for the trajectory stacking images;

$$\begin{aligned} I_T(u,v,1) &= d^x_{T+k-1}(p_k) \\ I_T(u,v,2) &= d^y_{T+k-1}(p_k) \\ u = [1:w], v &= [1,h], k = [1:L] \\ p_1 = (u,v), p_k &= p_{k-1} + d_{T+k-2}(p_{k-1}), k > 1 \end{aligned} \tag{2.10}$$

Where (p, d, t) represent given pixel point, displacement vector and sampled frame, respectively. Pk is the kth point along the trajectory, that starts from (u,v). As it is shown in the Figure 2.4, both methods create two channel outputs, but their calculation strategies are different. For optical flow stacking displacement vectors (d) are sampled at the same position for given frames. On the other hand, for the trajectory stacking displacement vectors are sampled along the trajectory. Displacement vectors and theirs corresponding frames are colorized with the same colour for clarification. At the end we have L frames with two channel showing the displacement either using optical flow or trajectory stacking.

2.2.2 Fusing Networks

In this sub-section we will share the results that Simonyan et al obtained from their work. Since we will be using similar approaches in our networks, effect of fusing strategy will be correlated. By reason of training two separate network, which isolates spatial and temporal learning process, Simonyan et al suggest fusing these networks in the end. They suggest two fusing options for the softmax score; either averaging or a linear SVM.

Finally we want to share the top results that Simonyan et al obtained for individual networks and fused network, to see the effect of fusion. For the Spatial ConvNet,

Table 2.1: Two Strem Network results given in ([1]).

Network	Accuracy
Spatial ConvNet	72.8%
Temporal ConvNet(Optical Flow Stacking)	81.2%
Temporal ConvNet(Trajectory Stacking)	80.2%
Two Stream ConvNet (Fusion method=averaging)	86.2%
Two Stream ConvNet (Fusion method=SVM)	87.0%

network is pre-trained on ILSVRC-2012 and fine-tuned on UCF-101. For Temporal ConvNet architectures, optical flow and trajectory stacking, 10 frame are used. We can see that, two stream configuration increases the accuracy.

2.3 CNN Architectures for Feature Extraction

In this sub-section, we will cover most well-known CNN architectures ([37], [38], [4], [2], [39], [5]). Ever since the rises of the deep neural networks, hand crafted features becoming rarer for image description. Given that there are deep architectures that are trained over large dataset such as ImageNet, their learned features are general enough for most of the tasks and they describe images better than most of the hand crafted features.

We will introduce VGG-Net, InceptionNet, ResNets and ResNext respectively. Each of these networks have introduce different approaches to overcome common problems such as vanishing gradients or increasing the depth without over fitting, controlling the number of parameters etc.

2.3.1 ImageNet Dataset

ImageNet ([40]) is one of the biggest dataset and it is used most of the challenges, so we want to mention it quickly. ImageNet, has over 15 million labelled high-resolution images with around 22,000 categories. ILSVRC (ImageNet Large Scale Visual Recognition Competition) uses a subset of ImageNet. It has 1000 categories

and contains around 1000 images in each. In all, there are roughly 1.3 million training images, 50,000 validation images and 100,000 testing images. Sample from the ImageNet is givin in 2.5.

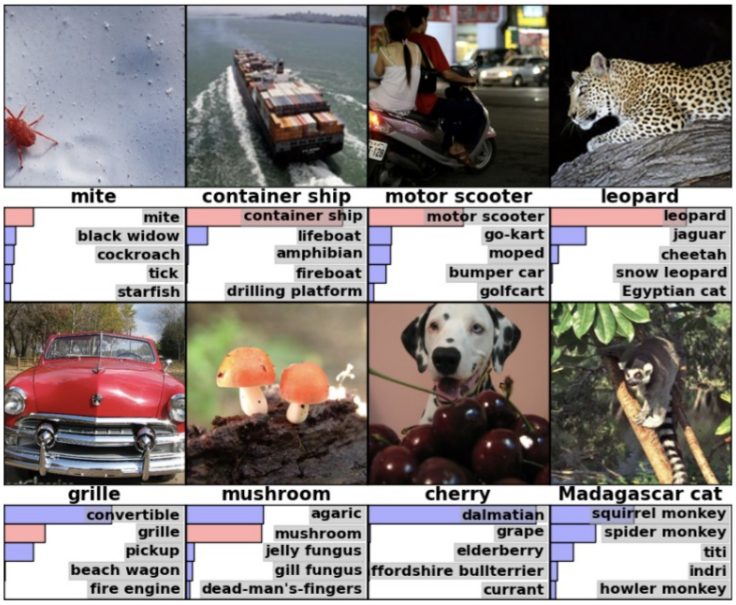


Figure 2.5: ILSVRC samples.

2.3.2 VGGNet Architecture

VGGNet was invented by Visual Geometry Group from University of Oxford [37], and it was the 1st runner-up, of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the classification task. Proposed architecture (VGG16) and tested networks are given in Figure 2.6.

In Figure 2.6 we can see that, VGG-Net uses smaller kernel compared to previous similar networks. 3x3 kernels were used for all the convolution operations which is the smallest size to capture the notion of left/right, up/down from the given pixel location. Previous networks uses bigger kernel for the first convolution layer, which drastically increases the number of parameters. To overcome this problem, these networks generally uses bigger stride sizes to reduce the output size of the convolution. Instead, VGG-Net uses smaller kernels with stride of 1 and keeps the input and output dimension is the same for the convolution layers. Sizes are only changed with

pooling layer.

For all the hidden layers, ReLu is used as activation function. VGG does not use Local Response Normalization (LRN). They have shown empirically, it does not improve accuracy.

VGG has three fully-connected layers; the first two have 4096 channels each and the third has 1000 channels. Features generally taken from second fully connected layer.

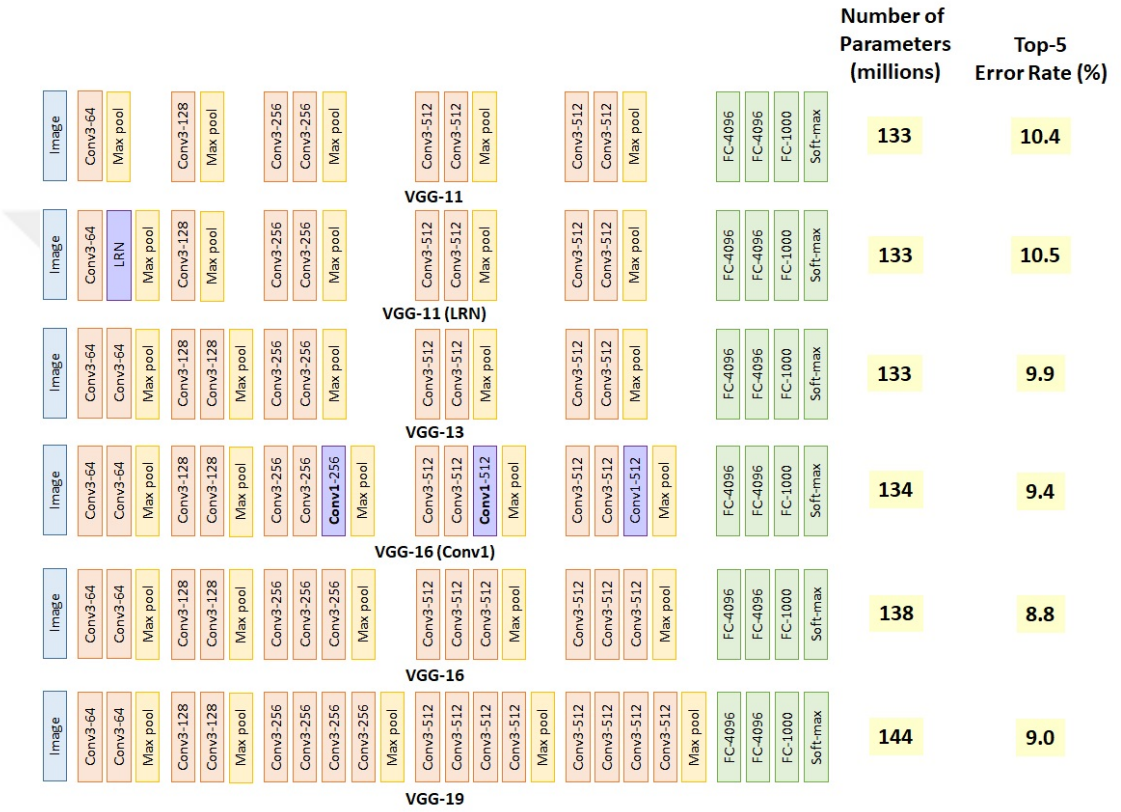


Figure 2.6: VGG-Net Architectures. Figure is taken from [10].

2.3.2.1 VGGNet

In this subsection, contributions of VGG architectures will be given. The most important feature of VGGNet is, it introduced the way of increasing the depth of the network without over fitting. To do so, they have used small convolution kernels (3x3), which reduce the parameters significantly compared the bigger counterparts. They have also shown that, learnt features are general enough to use as feature ex-

tractor without fine-tuning with the other datasets.

They stated that using 3x3 kernels give better results. Actually, using two successive 3x3 (with stride 1) convolution has the same effect of 5x5 convolution as it can be seen in the below figure. The advantage of this strategy is, it reduces total number of parameter while applying two times non-linearity for the given scenario, which makes the decision function more discriminative. Parameter reduction can be seen by simple calculation. Assume that we have C channel (number of filter for the given convolution), then for the 5x5 convolution we would have 5^2C^2 weights, while for the two time 3x3 convolution we would have $2(3^2C^2)$ weights, which is less than 5x5 case.

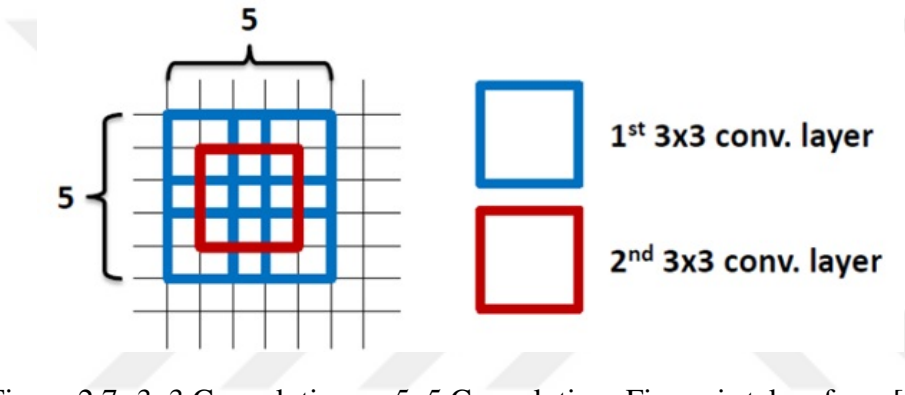


Figure 2.7: 3x3 Convolution vs 5x5 Convolution. Figure is taken from [10].

To investigate the effect of depth, they have created networks with same repeating architecture with different depths. They have created 4 network with different depth. When we look at the number of parameters versus error rates, the following conclusions can be drawn;

- For VGG-11 they have added 11 layer and used only 3x3 convolution and max pooling. This network obtains 10.4% error rate (similar to that of ZFNet in ILSVRC 2013).
- Additional to base architecture, for VGG-11 (LRN) they have added local response normalization (LRN) operation, which is suggested by AlexNet. This network obtains 10.5% error rate. So, LRN operation didn't increase the accuracy.

- For VGG-13, they have added 2 more layer and obtained 9.9% error rate, which means the additional convolution layers helps the classification accuracy.
- For VGG-16 (Conv1) they added three 1×1 convolution layers and obtains 9.4 % error rate. Additional 1×1 convolution layers helped to increase accuracy. For the VGG-16 they have changed these 1x1 convolution with 3x3 and further improved the results by obtaining 8.8 % error rate.
- Finally, for VGG-19 they have obtained 9.0% error rate which means the deep learning network is not improving by adding more number of layers.

VGG-Net showed us that increasing the depth with controlling the over fitting can increase the performance. They have created very deep networks (up to 19 layer) and archived state of art performance. VGG-Net is still used for feature extraction.

2.3.3 Inception Architectures

In this section we will talk about GoogleNet architectures. GoogleNet [2] first introduce in 2014 where it is the winner of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) in the classification task. Over the years there are some improvement for the baseline architecture. First we introduce the general idea of the GoogleNet, then we will give the improvements in the sub sections respectively.

When we look at the general trend of the CNNs to improve the performance, increasing the depth and number of kernel is the most popular approach. But increasing the depth typically means a larger number of parameters, which makes the network more vulnerable to over fitting. The other drawback of increasing the network size is the dramatically increased use of computational resources.

GoogleNet heavily influenced “Network in Network (NiN)” architecture, which is introduced by Lin et al [11]. Traditional CNN architectures use linear kernels for convolution followed by non-linear activation functions. But Lin et al suggest that, level of abstraction is low for these generalized linear models. Their suggestion is replacing linear models with nonlinear function approximator, which can increase the model quality. In NiN, linear model is replaced with a ”micro network” structure (multilayer

perceptron) which models the nonlinear function approximator. Suggested networks is given in Figure in 2.8.

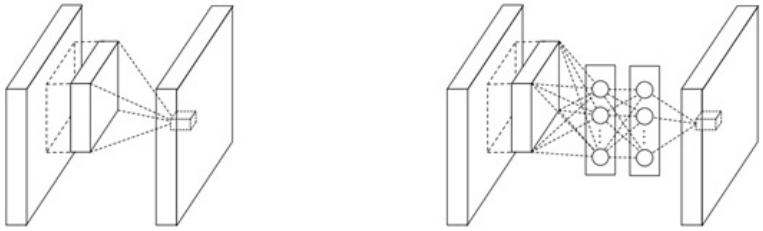


Figure 2.8: Proposed Network in Network architecture in [11].

Inception architecture has a different design compared to previous deep networks. Before Inception, previous trend has been to increase the number of layers, while using dropout to control over fitting. On the other hand, Inception uses similar idea to NiN. They have used 1x1, 3x3, 5x5 convolution and max pooling in parallel and merged the output of the filters of all those layers by concatenating into single output, which will be input of the next Inception module. In the figure 2.9, naive way of concatenating the outputs is depicted.

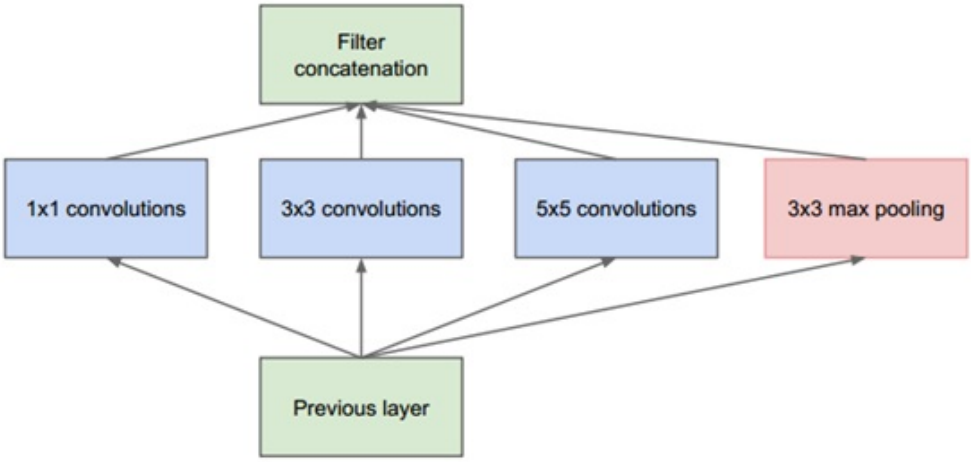


Figure 2.9: Naive Inception architecture given in [2].

Inception-V1 architecture is given in Table 2.2. When we look at the table 2.2, we can see that inception layers does not change the spatial dimension. So every convolution

operation and max pooling should produce the same spatial dimension with the input. Dimension reduction is occurred with occasional max-pooling layers with stride 2 to halve the resolution of the grid.

Table 2.2: Proposed Inception-V1 Network Architecture in [2].

Type	**Patch Size/ Stride**	**Output Size**
convolution	7×7/2	112×112×64
max pool	3×3/2	56×56×64
convolution	3×3/1	56×56×192
max pool	3×3/2	28×28×192
inception (3a)		28×28×256
inception (3b)		28×28×480
max pool	3×3/2	14×14×480
inception (4a)		14×14×512
inception (4b)		14×14×512
inception (4c)		14×14×512
inception (4d)		14×14×528
inception (4e)		14×14×832
max pool	3×3/2	7×7×832
inception (5a)		7×7×832
inception (5b)		7×7×1024
average pool	7×7/1	1×1×1024
Dropout (40%)		1×1×1024
linear		1×1×1000
softmax		1×1×1000

2.3.3.1 Inception-V1

The main contribution of the GoogleNet architectures is providing a solution for creating deeper networks while controlling the number of parameters. GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the AlexNet

(winner of 2012), while being significantly more accurate.

Before showing the way of reducing the computational blow up, first we want to show how to concatenate different kernel outputs. After calculating the output of each kernel and max pooling operation, we concatenate the responses along the depth as it can be seen in the figure 2.10.

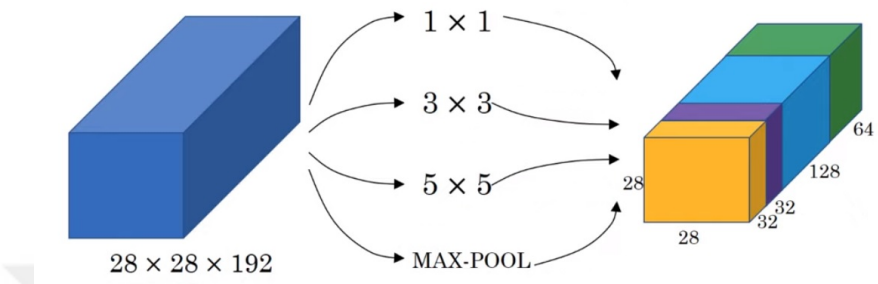


Figure 2.10: Concatenating the outputs. Figure is taken from [12].

The Problem of Computational Cost When we look at the output sizes of the Inception layers it starts with 256 (Inception 3a) and goes up to 1024 (Inception 5b). Considering the depth is too high, this will produce enormous numbers to compute.

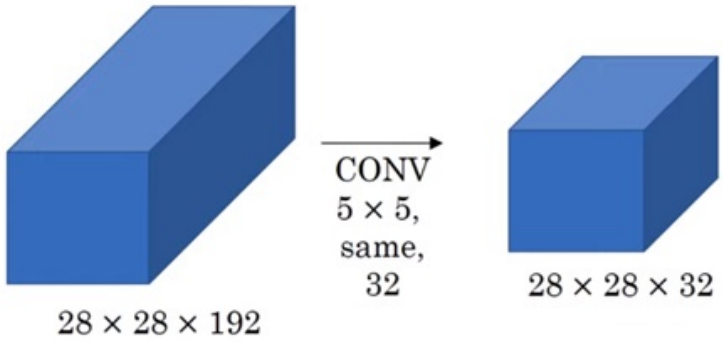


Figure 2.11: Computational cost without dimension reduction. Figure is taken from [12].

Let us investigate the input of the Inception 3a module. Here we have input size of

28×28×192. If we convolve it directly for the 5x5 kernel we would need [(5×5×192) x (28x28x32)] 120M calculation (2.11). We need to somehow reduce the depth before convolving the bigger kernels. Thus, 1×1 convolutions are used to reduce computation. We reduce the depth by embedding the information to lower dimension before the expensive 3×3 and 5×5 convolutions.

Embedding the lower dimension reduce the number of parameters significantly. This can be seen in figure 2.12, first we convolve the input with 1x1x192x16 (w,h,d,c) kernels. After this convolution we use (5x5x16x32) kernels. Notice that changing the depth of 5x5 kernels from 192 to 16 will reduce the computation drastically. Added sub layer is called as bottleneck layer, since it reduces the dimension. For the first part we would have [(1×1×192) x (28x28x16)] 2.4M and for the second part [(5×5×16) x (28x28x32)] 10M and in total we would do about 12.4M calculation, which is about 10% of the first calculation.

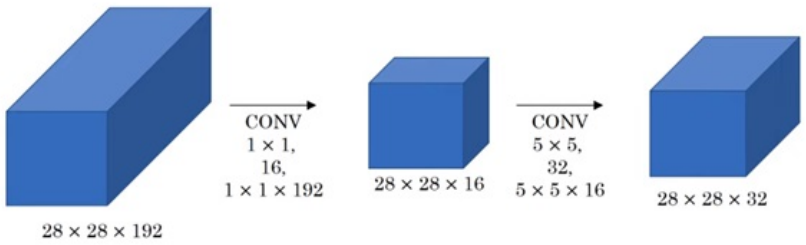


Figure 2.12: Bottleneck layer in the Inception architecture. Figure is taken from [12].

After adding the 1x1 convolutions to Inception layers, we will end up the final module as it is given in figure 2.13;

Propagating the Gradients It is hard to propagate the gradients such a deep networks, to overcome this problem GoogleNet uses auxiliary classifiers in the middle. This idea is a strategy against vanishing gradient problem. They have used two auxiliary classifiers (output of 4a and 4d layers). Their loss get added to the total loss during the training (with weighting).

Structure of this auxiliary networks are given in the paper as;

- An average pooling layer with 5×5 filter size and stride 3, resulting in an

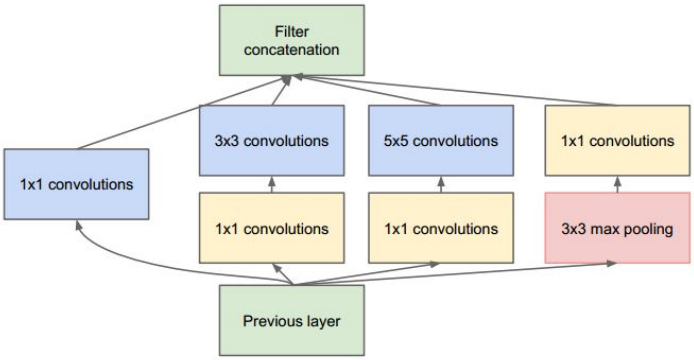


Figure 2.13: Final Inception module given in [2].

4×4×512 output for the (4a), and 4×4×528 for the (4d) stage.

- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.
- A linear layer with softmax loss as the classifier.

2.3.3.2 Inception-V2 and V3

In this sub-section we will introduce the improvements over the first Inception architecture. They have investigate the first architecture and made some important inferences [3], such as;

- Auxiliary classifiers do not contribute to increase the accuracy.
- Bottleneck layer should be avoided, since it causes lots of information loss. They have suggest different methods for dimension reduction instead.

They point out that we should use different Inception modules when we go deeper in the networks, since response of the network changes over depth.

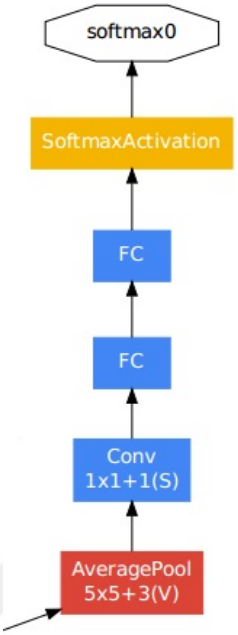


Figure 2.14: Auxiliary classifier proposed in [2].

To reduce the number of computation, they got inspired from VGG-Net. They have replaced 5x5 blocks with equivalent two successive 3x3 blocks. New Inception module given in figure 2.15;

This new module is used in the early layer of Inception-V2 network [3]. They have also further investigate the idea and suggest that we can also replace 3x3 convolution with 1x3 convolution followed by 3x1 convolution, which is 33% more efficient than 3x3 convolution. Authors find out that using this kind of asymmetric convolution gives great results for the mid layers, where grid sizes ranges between 12 and 20. This asymmetric convolution module is given in Figure 2.16 and final module is depicted in figure 2.17;

Final architecture of Inception-V2 is given in table 2.3.We can see that they have also used factorization for the 7x7 convolution, that are outside the Inception blocks, into three 3x3 convolutions.

Additional to this architecture, they have also tried different optimization techniques and find out that RMSProp gives better results compared to momentum optimizer.

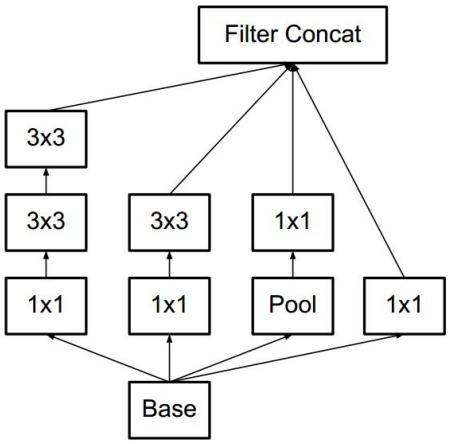


Figure 2.15: Proposed Inception module for early layers in [3].

They have also added batch normalization to auxiliary classifiers. This final version is called Inception-V3.

Table 2.3: Final Inception V2 architecture proposed in [3].

Type	**Patch Size/ Stride**	**Input Size**
convolution	3×3/2	299x299x3
convolution	3×3/1	149x149x32
convolution padded	3×3/1	147x147x32
max pool	3×3/2	147x147x64
convolution	3×3/1	73x73x64
convolution	3×3/2	71x71x80
convolution	3×3/1	35x35x192
3x Inception	Early Layer	35x35x288
5x Inception	Mid Layer	17x17x768
2x Inception	Final Layer	8x8x1280
max pool	8x8	8x8x2048
linear	Logist	1x1x2048
softmax	Classsifier	1x1x1000

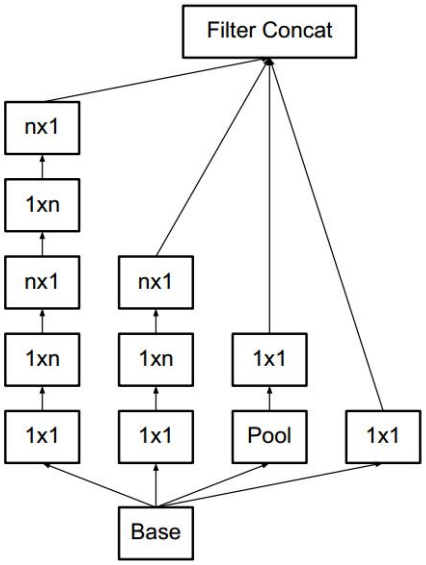


Figure 2.16: Proposed Inception module for mid layers in [3].

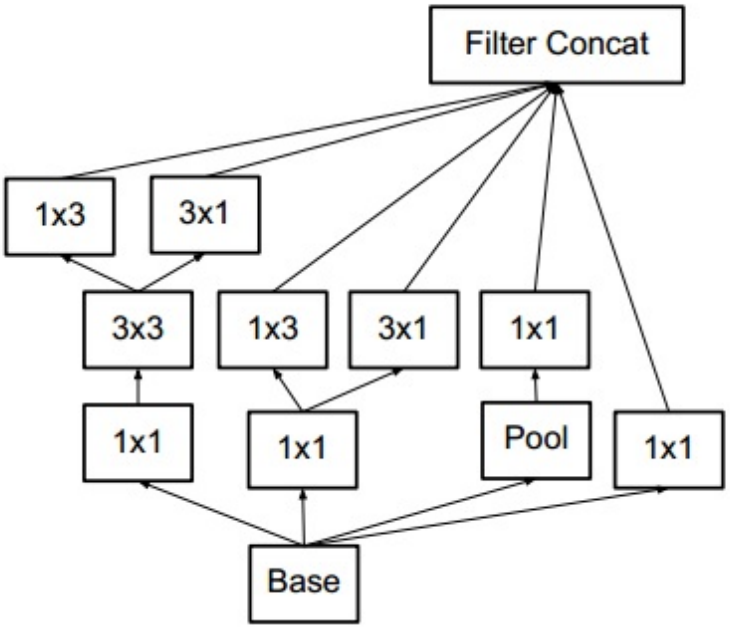


Figure 2.17: Proposed Inception module for final layers in [3].

2.3.4 Residual Connection Based Architectures

Neural networks becoming more and more deeps over time. There are lots of evidence showing that depth is crucial parameter for learning higher level features. General design pattern to increase the depth is stacking similar layers one after another. But interesting problem has been exposed; after some point stacking more layer does not improve the accuracy. On the contrary, accuracy saturated at some point and then starts to degrade rapidly.

First, this might be seen as vanishing gradient problem, but authors [4] stayed that after some experiments it turns out gradients are healthy. It is shown experimentally; after certain point, adding more layer increase the training error. This phenomena is called "degradation problem". Experimental result is shown in Figure 2.18. We can see that, 56-layer network has more error compared to 20- layer network for both training and test sets.

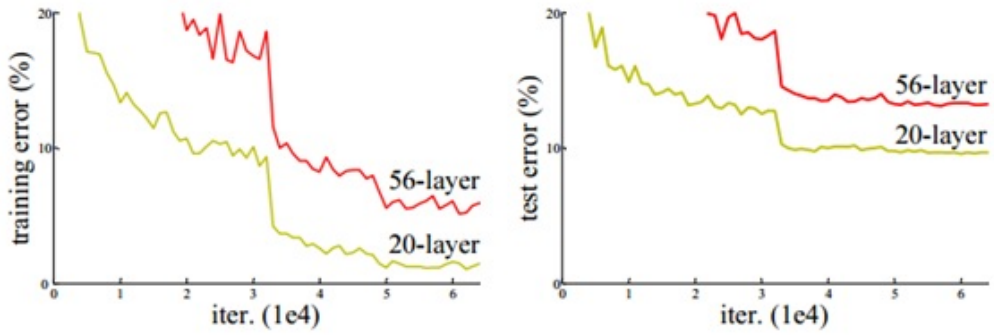


Figure 2.18: Degredation graph for deep networks. Figure is taken from [4].

ReseNet uses shortcut connection as residual function to solve the degradation problem. It contains identity function for its every layers. By doing so, it can be guaranteed that in the worst case newly added layer should give similar result by setting the layer as identity function. The idea is given in Figure 2.20 on the right. 34-layer version of ResNet is given in figure 2.19.

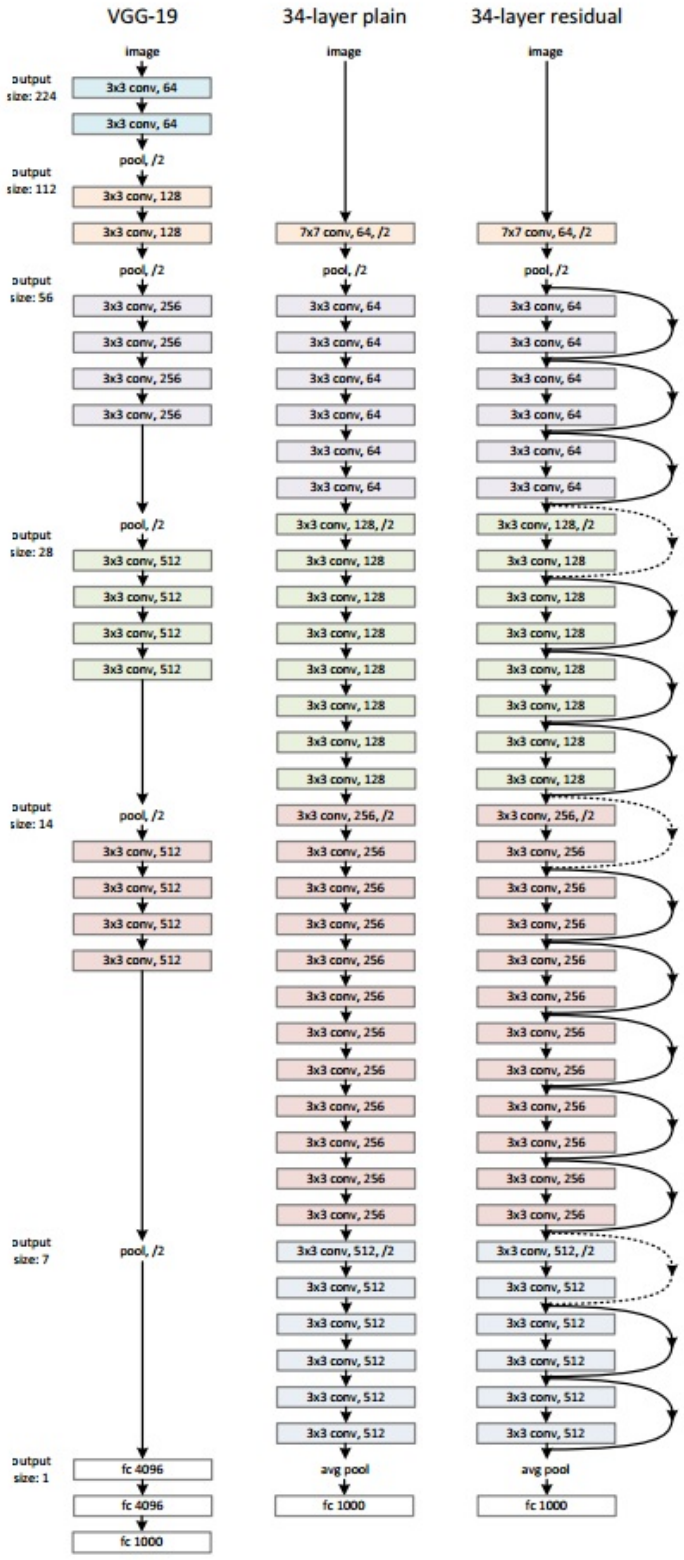


Figure 2.19: Proposed ResNet architecture in [4].

2.3.4.1 ResNet

Let us say that F(x) is the mapping function of the few stacked layers. If we assume that our network would eventually approximate the desired solution of F(x), then we could also say that we can approximate F(x) + x, the residual function (the idea is depicted in Figure 2.20 on the right). By doing that, instead of approximating the F(x), residual function (H(x) = F(x) + x) can be approximated. Main advantages of using residual function is, if the newly added module produces worse results than we can simply set F(x) =0 and create an identity mapping. This reformulation does not have any computational burden, it's just a simple addition. The dimensions of x and F must be equal. When we reduce the spatial dimension in F(x), we should add a linear projection Ws to match the dimensions:

$$y = F(x, \{W_i\}) + W_s x \tag{2.11}$$

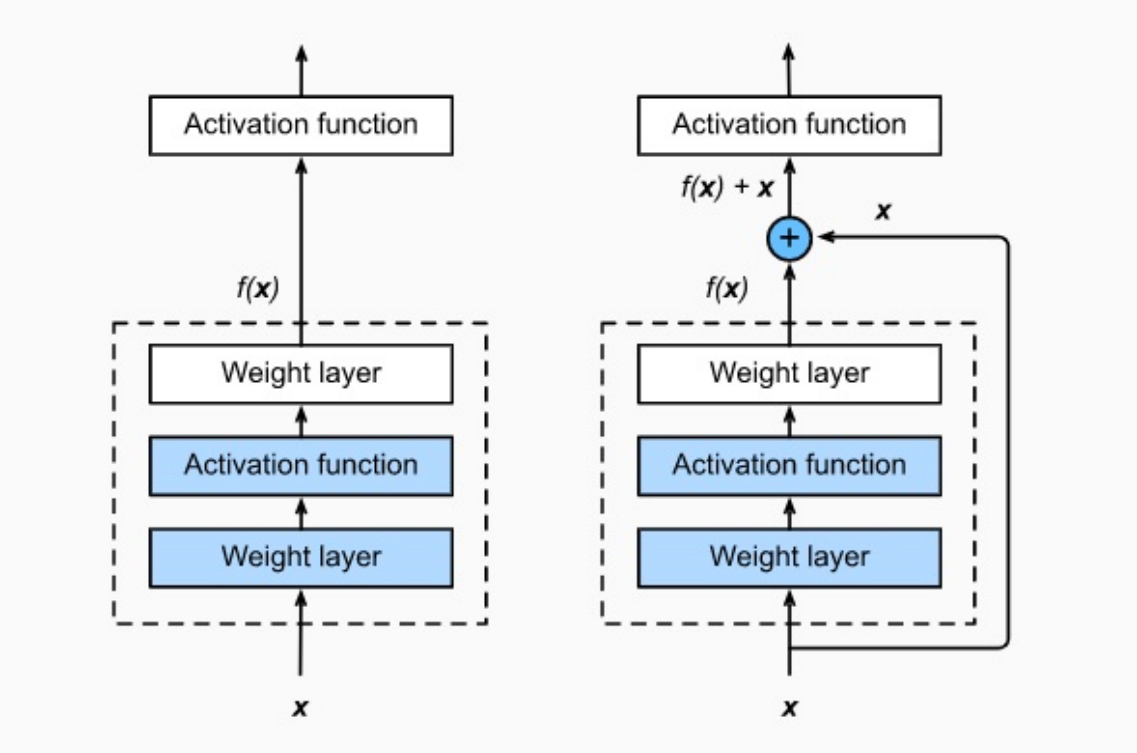


Figure 2.20: ResNet Block. Figure is taken from [13].

Authors tested different models to show the effect of shortcut connections. They

have created two similar network, one with shortcut connections and other with a traditional network model. They have followed similar design paradigm to VGG-Net. The convolutional layers mostly have 3×3 filters. For the same input/output size they have used same the number of filters. If the spatial dimension of the output is halved, then they have increased the number of filters twice. This approach will kept same the time complexity per layer.

Results are given in Table 2.4 for 34 layer and 18 layer models. When we look at the results we can see that the deeper 34-layer plain net has higher validation error than the shallower 18-layer plain net, which is expected since the degradation problem. For the ResNet counterparts we can see that 34-layer ResNet is better than the 18-layer ResNet. Which indicates that ResNets solve the degradation problem.

Table 2.4: Resnet results with plain networks [4].

Validation Errors	**Plain**	**ResNet**
18 Layers	27.94	27.88
34 Layers	28.54	25.03

We can further say that, 34 layer plain network has the highest error and even 18 layer ResNet achieves better performance than both 34 layer and 18 layer plain counterparts. Error rates of all networks are given in Figure 2.21.

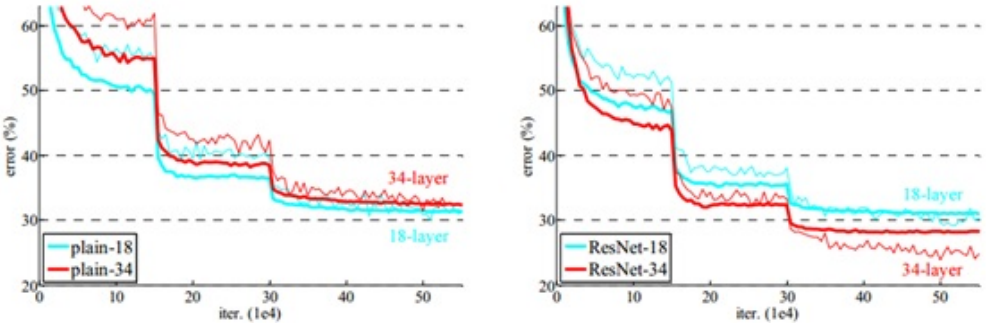


Figure 2.21: Resnet training graph. Figure is taken from [4].

2.3.4.2 ResNext

After the success of ResNet and Inception architectures, ResNext somehow combines the powerful parts of both architectures. ResNext uses residual function like ResNet to prevent degradation. Also uses sub-modules like Inception, calculate the response of each operation and concatenates the outputs at the end of the each module. Here they have introduced these sub-operations inside the given module as a dimension called “cardinality” [5].

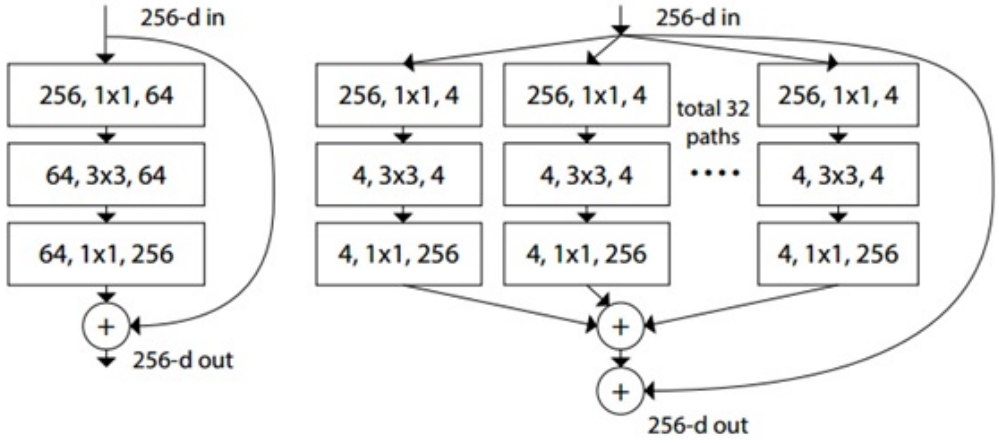


Figure 2.22: Proposed ResNext module in [5].

Similar to ResNext, Inception-ResNet also uses this kind of design pattern. But ResNext further simplify the design of each module by utilizing the same sub-operations along the lower-dimensional embedding (along the cardinality dimension). Their primary concern is to reduce to total number of hyper parameters.

Overall, they have combined the strong parts of VGGNet and Inception architectures by using small kernels with repeating layers like VGGNet and using mini networks like Inception. For the same input/output size they have used same the number of filters. If the spatial dimension of the output is halved, then they have increased the number of filters twice. This approach will kept same the time complexity per layer.

They have also stated that cardinality dimension is very important to increase the accuracy and showed that increasing cardinality is more effective compared to creating

deeper networks.

Aggregated Transformations As already mentioned, ResNext uses set of parallel transformations like Inception architecture, which is called cardinality. They have liken this operations as a simple neuron operation, which also performs splitting, transforming, and aggregating the given input. This inner product operation performed by neuron could be changed any transformation. By setting this transformation as a set of convolution operations we could create a network. Conversely to "Network-in-Network" strategy which increase the dimension of depth, this model may be called "Network-in-Neuron" which creates a new dimension.

$$F(x) = \sum_{i=1}^{C} T_i(x) \tag{2.12}$$

Ti projects x into a lower (or equal) dimensional space. Each transformations are aggregated in the end. Authors used a bottleneck layer to create low dimensional embedding (1x1 layer). Aggregated transformation is used as residual function, overall input/output relation is given as [5];

$$y = x + \sum_{i=1}^{C} T_i(x) \tag{2.13}$$

ResNext achieves better results compared to ResNet, while keeping the number of operations nearly the same. For a given block in figure 2.22, we can see the number of parameters in table 2.6. As we can see, ResNext reduces the number of parameters in the mid layer with using low level embedding, by that way it keeps the total number of parameters nearly the same.

Cardinality vs Depth To keep the overall complexity same, we should reduce the depth while we are increasing the cardinality. In figure 2.22, a layer is denoted as number of input channels (depth), filter size, number of output channels (depth). Given architecture reduces the depth to 4 while using cardinality as 32. Alternative setting could also be utilized, in table 2.7 we can see the results. We can say that

Table 2.5: ResNet and ResNext architectures. Table is taken from [5].

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		1×1, 64 3×3, 64 x 3 1×1, 256	1×1, 128 3×3, 128, x 3 (C=32) 1×1, 256
conv3	28×28	1×1, 128 3×3, 128 x 4 1×1, 512	1×1, 256 3×3, 256, x 4 (C=32) 1×1, 512
conv4	14×14	1×1, 256 3×3, 256 x 6 1×1, 1024	1×1, 512 3×3, 512, x 6 (C=32) 1×1, 1024
conv5	7×7	1×1, 512 3×3, 512 x 3 1×1, 2048	1×1, 1024 3×3, 1024, x 3 (C=32) 1×1, 2048
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5×**10^6	**25.0×**10^6
FLOPs		**4.1×**10^9	**4.2×**10^9

Table 2.6: Number of parameters of ResNet and ResNext modules.

# of parameters	**ResNet**	**ResNext**
First layer	256x64 (16384)	256x4x32 (32768)
Second layer	64x3x3x64 (36384)	4x4x3x3x32 (4608)
Last layer	64x256 (16384)	4x256x32 (32768)

increasing the cardinality while reducing the depth gives better results.

Table 2.7: Cardinality vs depth results. Table is taken from [5].

	Setting	Top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	22.2
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	21.2

2.3.5 Depth-wise Convolution Based Architectures

In this subsection, we investigate architectures that are created based on depth-wise separable convolution idea. MobileNet and Xception architectures will be introduced. A standard convolution layer takes a future image and produce an output. For this calculation N kernels (W_kxH_kxD_k) are used, where (W_kxH_k are the width and height of the filter and D_k is the depth of the feature image (also kernel depth). Traditional convolution calculates the cross-channel correlations and spatial correlations simultaneously. To calculate the response of a single pixel we need to do W_kxH_kxD_kxN calculations. For a given feature image with a spatial dimensions of W_fxH_f, number of calculation for each convolution;

$$W_k H_k D_k N W_f H_f \tag{2.14}$$

To reduce the number of calculation, depth-wise separable calculation can be used. When the output is calculated in this way, convolution operation is factorized in to two parts; depth-wise convolution which applies single filter to each input channel and point-wise calculation (1x1) which combines the outputs of the depth-wise convolutions. When we calculated the output response of the calculation, for the depth-wise

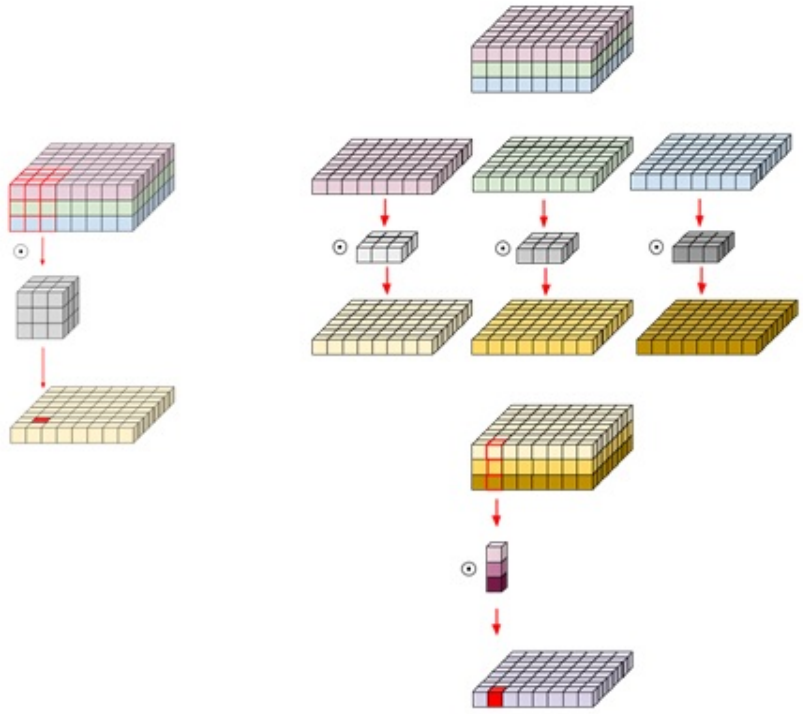


Figure 2.23: Standard Convolution vs Depth-wise separable convolution. Figure is taken from [14]

convolution we would have;

$$W_k H_k D_k W_f H_f \tag{2.15}$$

number of calculations. For the point-wise convolution part, since the kernel size is 1, we would have;

$$D_k N W_f H_f \tag{2.16}$$

number of calculations. So, the total cost of depth-wise separable convolution can be given as;

$$W_k H_k D_k W_f H_f + D_k N W_f H_f \tag{2.17}$$

So to calculate the reduction ratio in the computation, compared to traditional convo-

lution, we can take the ratio of previous equations which gives;

$$1/N + 1/W_k H_k \tag{2.18}$$

2.3.5.1 MobileNet

The MobileNet is created based on depth-wise separable convolution idea [6], [41]. Architecture of the networks is given table 2.8. It contains 28 layers, when we count depth-wise and point-wise convolutions separately. Additional to depth-wise separable convolution idea, MobileNet introduces two hyper parameters to control the network parameters, that are width and resolution multiplayers. The idea is to create adaptable network that can run at different devices effectively.

Width Multiplier: Thinner Models This parameters (α) controls the depth of the network, where (α) $\in$ (0; 1]. By changing the α value, number of filters at each layers and depth of the feature channels could be controlled. For a given α value, D_k (the depth of the feature image) becomes αD_k and N (number of kernel) becomes αN. So, the total cost of depth-wise separable convolution with α parameter can be given as;

$$W_k H_k \alpha D_k W_f H_f + \alpha D_k \alpha N W_f H_f \tag{2.19}$$

Resolution Multiplier: Reduced Representation This parameters (ρ) controls the spatial resolution of the network, where $\rho \in (0; 1]$. ρ parameter is multiplied width and height of the each feature channels. So, the total cost of depth-wise separable convolution with α and ρ parameters can be given as;

$$W_k H_k \alpha D_k \rho W_f \rho H_f + \alpha D_k \alpha N \rho W_f \rho H_f \tag{2.20}$$

Table 2.8: MobileNet architecture. Table is taken from [6]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
Conv dw / s1 (5x)	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1 (5x)	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

2.3.5.2 Xception

Another architecture created based on depth-wise separable convolution is Xception. The idea comes from creating uniform version of Inception-V3 module. As we discussed in section 2.3.3.2, Inception module uses 1x1 convolution, followed by 3x3 convolution. However, Inception-V3 module contains different filters and layers in-

side the Inception module. Here, authors used uniform design pattern and used 1x1 convolution followed by 3x3 convolution (2.24).

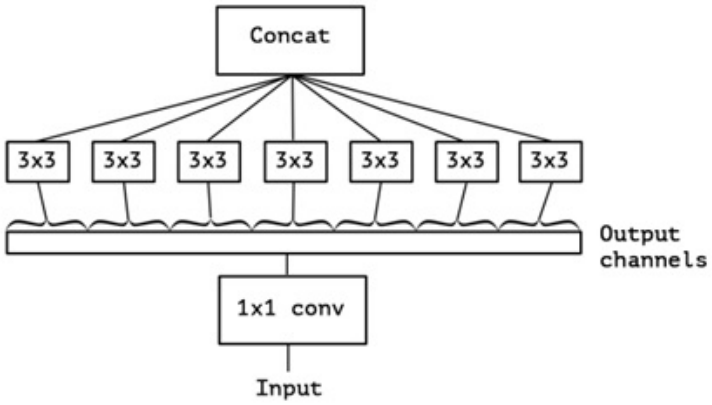


Figure 2.24: Xception Module. [15]

Main difference, compared to depth-wise separable convolution, is the order of operations. Here, 1x1 convolution is applied first. As it can be seen in the figure, Xception architecture is a linear stack of depth-wise separable convolution layers with residual connections.

2.4 Attention Models

In this chapter we would like to introduce attention models. Attention mechanism first introduce in "Transformer networks" [42] for encoder-decoder architectures. Back then its popularity increases in natural language processing (NLP) and machine translation communities. For the computer vision application it is also getting popular recently.

For any machine learning application, we would like to find a good representation for the given input. But sometimes extracted features have lots of redundant data and somehow we want to find the features that give the most salient representation for a given task. So, we need a mechanism that performs feature recalibration, such that networks can learn to select most informative features and suppress the others that are less useful. Attention mechanisms are exactly used for this tasks.

There are two different attention mechanism, hard attention and soft attention. In essence, attention reweighs certain features of the network according to some externally or internally (self-attention) supplied weights. Hereby, soft attention allows these weights to be continuous while hard attention requires them to be binary, i.e. 0 or 1. The main drawback of hard attention is that it is not differentiable. To train the attention gates, we have to use sigmoid or softmax functions (soft attention).

2.4.1 Squeeze and Excitation Networks

In [16] Hu et al. introduced squeeze and excitation networks. In order to pick informative characteristics and suppress less informative ones, SE blocks may learn global information. SE block contains two successive blocks namely squeeze and excitation blocks. SE block structure is very simple and easy to add any state of art model (like ResNet, Inception or ResNext). It is also computationally very light; it contains channel wise addition and two activation function (ReLu and sigmoid).

SE block is given in figure 2.25;

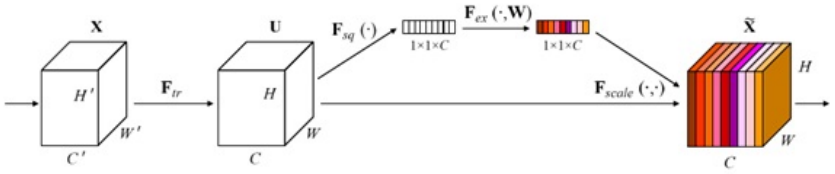


Figure 2.25: Proposed Squeeze and Excitation Block in [16].

For any transformation from input X to output U (e.g. convolution), we would like to learn channel wise relation. But while we are convolving different kernels with the input, output responses are independent from each other. To overcome this problem, authors introduce squeeze operation. Squeeze operation averages the channel wise information. This can be seen as average pooling operation with filter size is equal to output image size. For each channel squeeze operation defined as [16];

$$Z_C = F_{sq}(u_c) = \frac{1}{HxW} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j) \tag{2.21}$$

Here Uc is the output of the each convolution. Authors used average pooling for squeezing channel wise information, but it is possible to use different operators as well. They reported that average pooling gives slightly better result compared to max pooling.

After squeezing operation excitation operator follows. This operator aims to capture channel wise dependencies. To learn this kind of information, our operator should be general enough and to be able to learn non-linear function. It also needs to be a simple function to control its response. Authors suggest following gating operation to meet the desired response;

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1 z)) \tag{2.22}$$

The excitation multiplies the output of the squeeze block by W1. Then output is given to ReLu unit. After ReLu operation output again multiplied another set of weights W2 and given to sigmoid function at the end. W1 reduces the dimensionality by the factor r, whereas W2 increases it again to the original number of channels.

Finally, the channel feature responses of Ftr are multiplied by the weights obtained from the excitation block. This can be viewed as a self-attention function on the channels using global information.

$$\tilde{X}_c = F_{scale}(u_c, s_c) = s_c u_c \tag{2.23}$$

Excitation operation maps the input specific descriptor z to a set of channel weights. By that way we learn set of weights that are depend on channel aggregation response, so these weights are depend on each other. This operation captures channel wise dependencies.

Output of the excitation operation is a vector of length C (number of channel). This vector, obtained from the output of the excitation block, is multiplied by the channel feature responses of Ftr. This output represents the output of convolution operation modified by channel dependencies.

When we look at the effect of SE-block at the different depth, its role changes through

the network. For the shallow layers, where we have more general features, SE-block amplify more informative part of class independent features. When we go deeper its effect becomes increasingly specialised for each class. Since this is the general feature evaluation along the depth, this kind of effect is predictable.

2.4.1.1 Implementation to State of Art Networks

In this subsection, we have introduce how to apply SE-blocks to state of art networks, which are discussed previous sections. SE block can be directly integrated for any standard CNN architecture by inserting it after each convolution operation (nonlinearity operation is also applied before SE-block).

Here we will discuss how to apply for Inception and ResNet architectures. In figure 2.26 general implementation is given for both networks. Global pooling layer stands for squeeze operation. Following FC and ReLu operations is for dimensionality reduction. Final FC and sigmoid operations for increasing it again to the original number of channels. Output of this operation is given to scaling to get the output.

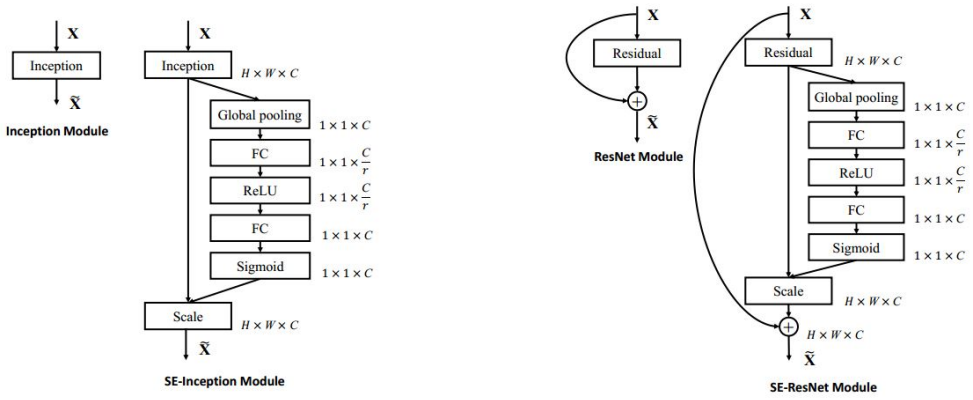


Figure 2.26: Implementation of SE Block for Inception and ResNet architectures. Figure is taken from [16].

Lastly in figure 2.27, it is given the shapes of SE-ResNet and SE-ResNext networks with traditional ResNet architecture. ResNext network is built with 32 cardinality-4 channel configuration. The inner brackets following by fc indicates the output dimension of the two fully connected layers in an SE module. So, dimension is reduced to

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d fc, softmax		

Figure 2.27: ResNet, SE-ResNet, SE-ResNext Architectures Comparison [16].

16 first then it is scaled back to original number of channels.

2.5 Sequential Learning for Time Series Inputs

This chapter will introduce sequential learning algorithms. So far we have investigated feature descriptors, which works on single frame. To create a correlation between successive frames, we need a sequential learning model. As a discriminative model, Recurrent Neural Networks and their special versions Long Short Term Memory units will be introduced.

2.5.1 Recurrent Neural Networks (RNN)

As a discriminative model, we will introduce Recurrent Neural Networks. The principle behind RNNs is to make use of sequential information. We assume all inputs (and outputs) are independent of each other in a typical neural network. While classifying a video stream, we need to know the information about previous frames as well as current frame. Our CNN architectures are not capable of storing this kind of information. Information about the previous outputs should be taken as input along with the current information.

This is the general idea behind the recurrent neural networks, which is depicted in

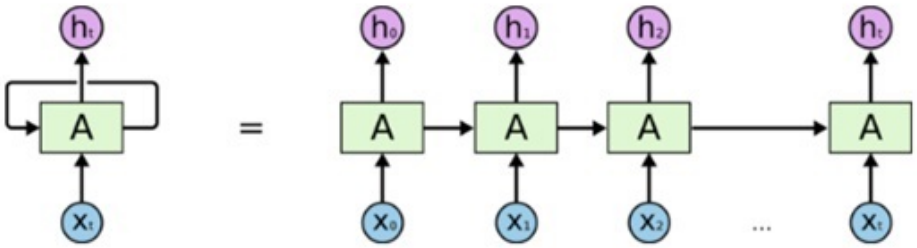


Figure 2.28: An unrolled RNN. Figure is taken from [17]

Figure 2.28. RNNs are cascaded networks one after another. For every element of a sequence, RNNs perform the same assignment. Each output is depend on previous calculations, which creates a memory.

Different types of RNNs In figure 2.29, different usage examples are depicted. One to one is just the classic neural network. One to many architecture could be used for image description where input is the given image and the outputs are the words that describes the scene. For action classification problem, like our case, many to one architecture could be used. This time each input is the input video frames, and the output in the final layer is the classification result. Lastly, many to many architectures generally used for language translation problems.

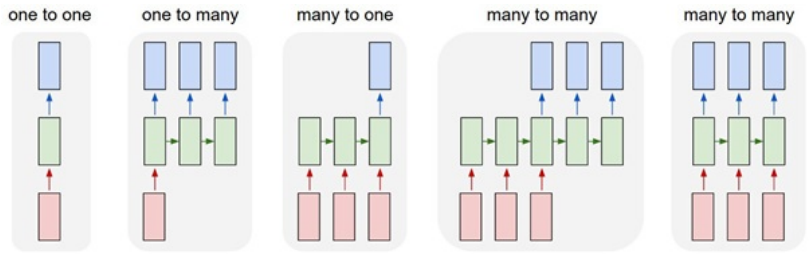


Figure 2.29: Different types of RNNs. Image is taken from [18].

Mathematical model In this subsection, we will be looking at the mathematical model. Here, W_{hh} weights between hidden states, W_{hx} is the weight at current input state. O_t is the current state output, h_t and h_{t-1} are the current and previous time

stamps. For a given state, relation between previous times stamp and input, given in current time stamp, could be written as a function;

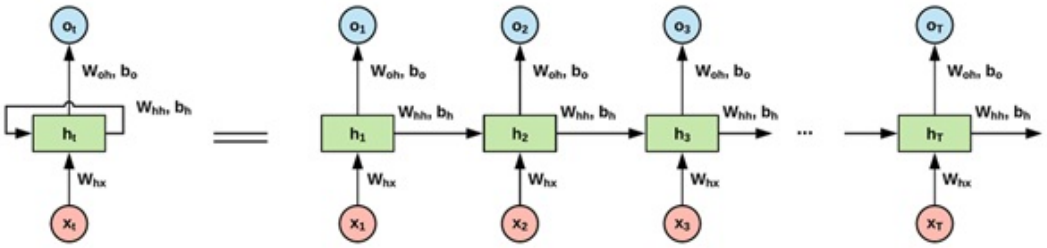


Figure 2.30: Mathematical model of RNNs.

$$h_t = f(h_{t-1}, x_t) \tag{2.24}$$

Here f could be any desired activation function, for example if we use tanh then h_t could be written as;

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \tag{2.25}$$

To propagate the error we should consider the error for both current and previous states, since error depends both of them. In figure 2.30 we can see the unrolled RNN at all time stamps.

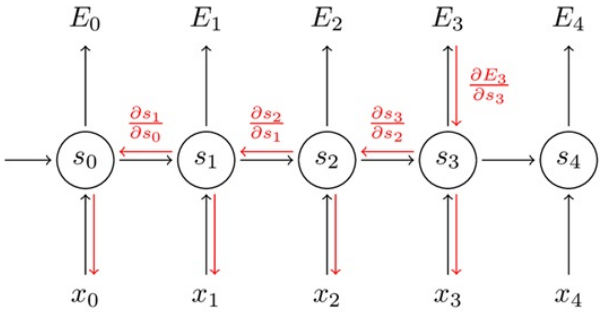


Figure 2.31: Backpropogate Through Time. Image is taken from [19]

Using chain rule we can write the error for the given unit;

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W} \tag{2.26}$$

But as it is given above h_t depends on previous states h_{t-1}, which is also depends on h_{t-2} and so on. So we shall write as;

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t}\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W} \tag{2.27}$$

So we need to go back into each time stamp and modify the weights to update the given networks, which is very expensive. Another problem with this setup is, we are propagating the gradients for lots of units, which creates vanishing gradient problem.

2.5.1.1 Long Short Term Memories (LSTM)

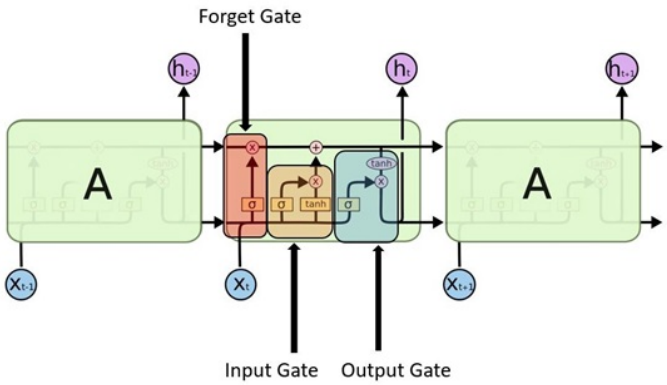


Figure 2.32: LSTM Unit. Figure is taken from [17].

LSTMs are one of the best solution for remembering long period information [43]. As a special type of RNN, LSTM is able to learn long-term dependencies. It has three gating operation; forget gate, input gate and output gate. While forget gate controlling the previous layer‘s contribution to the output, input gate controls the current state‘s contribution. By that way long period information could be remembered without

vanishing gradient problem. In the below formulations C_t represents the information that pass through the states, x_t represents current input and h_t represents the hidden states.

Forget Gate: Information comes from previous layers is processed by special gating operator, which is called forget gate. This gate decides how much of the past should be remembered. This decision is made by a sigmoid function, sigmoid takes the weighted sum of current and previous states and output a value between 0 to 1. While 0 means omit the given information, 1 means keep the information.

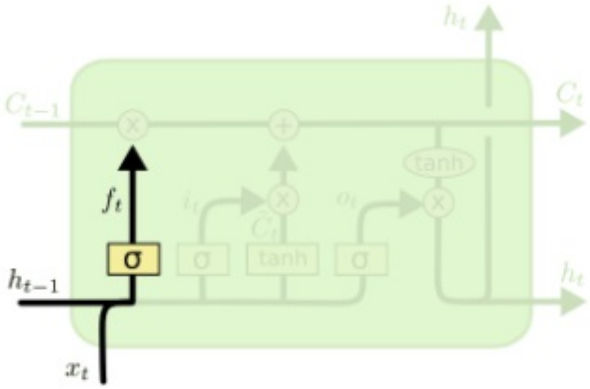


Figure 2.33: Forget Gate. Figure is adapted from [17].

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ f_t &= \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f) \end{aligned} \tag{2.28}$$

Update Gate/input gate: Similar to forget gate, input gate adjust the contribution of the new information. Additional to sigmoid, it also uses a tanh function. Information pass through the sigmoid function further weighted by a tanh function which gives output between $[-1, 1]$.

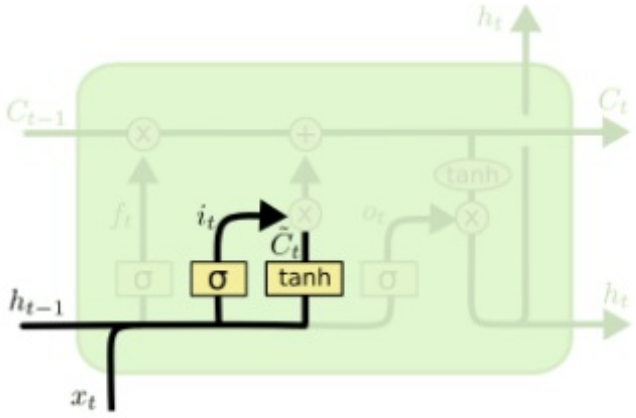


Figure 2.34: Input gate. Figure is adapted from [17].

$$
\begin{aligned}
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
i_t &= \sigma(W_{hi}h_{t-1} + W_{xi}x_t + b_i) \\
\bar{C}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\
\bar{C}_t &= \tanh(W_{hc}h_{t-1} + W_{xc}x_t + b_c)
\end{aligned}
\tag{2.29}
$$

Output Gate: Finally next information content is determined by aggregating the input and forget gate results. Next hidden state is also determined by using current input x_t, previous hidden states h_{t-1} and current information C_t. Sigmoid function decides which values to let through and tanh function gives weightage to the values which are passed deciding their level of importance ranging from $[-1, 1]$ and multiplied with output of sigmoid.

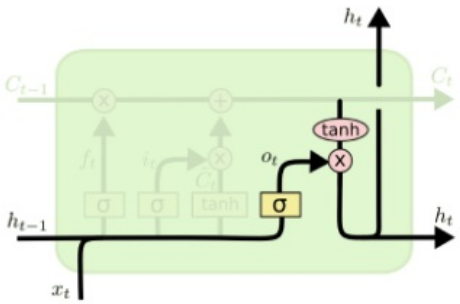


Figure 2.35: Output Gate. Figure is adapted from [17].

$$
\begin{aligned}
C_t &= C_{t-1} \odot f_t + i_t \odot \bar{C}_t \\
o_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_{hi} h_{t-1} + W_{xi} x_t + b_i) \\
h_t &= \tanh(C_t) \odot o_t
\end{aligned}
\tag{2.30}
$$

2.6 3D-CNN Models

Instead of using a separate feature extractor with a sequential learning algorithm, another popular solution is using 3D CNNs [32]. 3D CNNs are more suitable compared to 2D variants for spatiotemporal feature learning. While 2D CNNs are good for single image feature extraction, due to the absence of motion modelling, they are not completely suitable for videos.

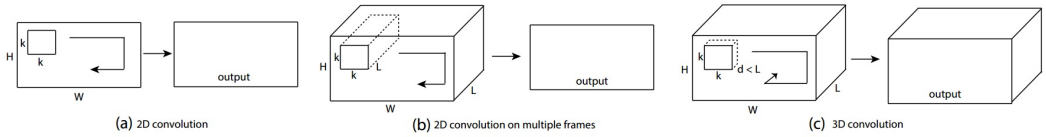


Figure 2.36: 2D vs 3D Convolution Results [20].

In 3D CNNs, convolution and pooling operations are performed spatio temporally while in 2D CNNs they are done only spatially. While 2D convolution and stacked 2D convolution outputting an image, 3D convolution outputs a volume. 2D stacked convolution here means that 2D convolution is applied to multiple images and treated as distinct channels. As a result both 2D and 2D stacked convolution lose temporal information of video; this is depicted in Figure 2.36.

In 3D CNN, the 3-dimensional kernels are used. By using 3D convolutions, 3D-CNN model extracts features from both spatial and temporal dimensions, hence capturing the motion information encoded in several adjacent frames.

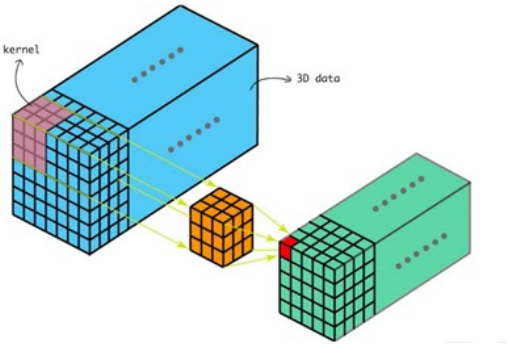


Figure 2.37: 3D Convolution.

2.6.1 I3D Models

Major problem with the 3D CNNS is initializing of the parameters. While we can use the pre-trained weights on ImageNet for 2D CNN models, for the 3D CNNs, training starts from random initializations, which affect the overall performance. ImageNet weights give good starting point, which have a powerful representation for the spatial features. Deep models that are trained on large datasets can be used as feature extractor for other tasks, since they are general enough for feature representation.

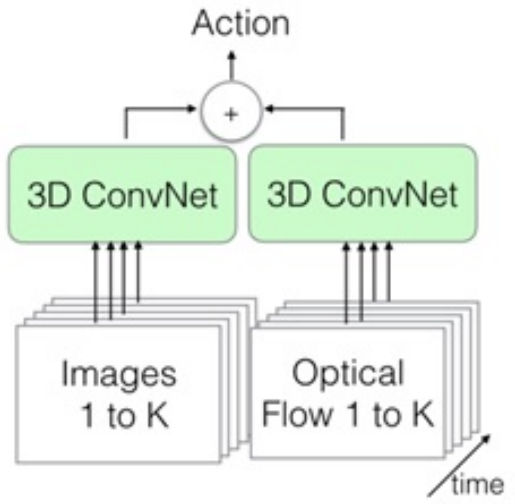


Figure 2.38: Proposed I3D model in [21].

The idea behind the I3D [21] is using the well-known 2D architectures by inflating the networks in 3D. With this kind of usage we can use pre-trained 2D weights in 3D convolution. They have also used two stream configuration here; one stream is

trained with RGB images and the other is trained with flow images. Final result is obtained by averaging the outputs. This model is called "Two Stream Inflated 3D ConvNets (I3D)". They have used Inception V1 architecture and obtain a very good performance boost.

Number of parameters: One of the drawbacks of the 3D-CNN models is number of parameters. Compared to other models, 3D CNNs have a lot more parameters due to the depth of the kernels. When we look at the 3D models, they have more parameters than the CNN-LSTM or two stream networks. But for the Two-Stream I3D, since they have used Inception Net, they have managed to control to number of parameters. As discussed in the previous chapters, Inception Net achieves good performance using less parameters compared to other models, this is also true for the inflated case.

2.7 Tandem and Hybrid Approaches for Learning Process

While we are using different algorithms in cascade, we have two alternatives. We can either train the whole model at once in an end to end objective, which is called hybrid approach, or we may train each sub unit separately and then create a pipeline by adding one after another, which is called tandem approach.

For tandem approach CNN is used as a feature extractor rather than as classifier. Extracted features then fed to sequential learning model. For the hybrid approach both features and RNN parameters are learnt iteratively. The main idea of hybrid approach is to alternatively update CNN and RNN parameters.

2.8 Proposed Model - Two Stream SE-ResNext50-LSTM Module

In this subsection, we will introduce our model. In the proposed method, we used CNN-LSTM networks with Two Stream configuration. For the feature extraction process, ResNext architecture is used with Squeeze and Excitation Network module. Then extracted features are fed to LSTM units.

Compared to similar ideas [34], we used hybrid architecture for both RGB and optical flow inputs. With this configuration, spatial network is trained with RGB inputs. This module can learn spatial features from RGB images. Furthermore, with the help of LSTM modules, it can also learn temporal features. For the temporal network, optical flow images are used. Again, CNN-LSTM hybrid architecture is used. If we use only CNN architecture for this network, network cannot learn long term dependencies, since sequential frames are used while calculating optical flow. Adding LSTM layers to the temporal network helps the network to learn long term dependencies.

In figure 2.39, proposed method is depicted. Here, both networks use SE-ResNext-LSTM architecture. First network is trained with RGB images and second is trained with flow images. Networks are trained separately. To calculate the output, we can calculate each networks outputs and take the average.

For the proposed architecture, features are extracted at the top layer of SE-ResNext-50 network. We used pre-trained ImageNet weights. After extracting the features, we pass the features to LSTM module. Training details will be given in section 3.

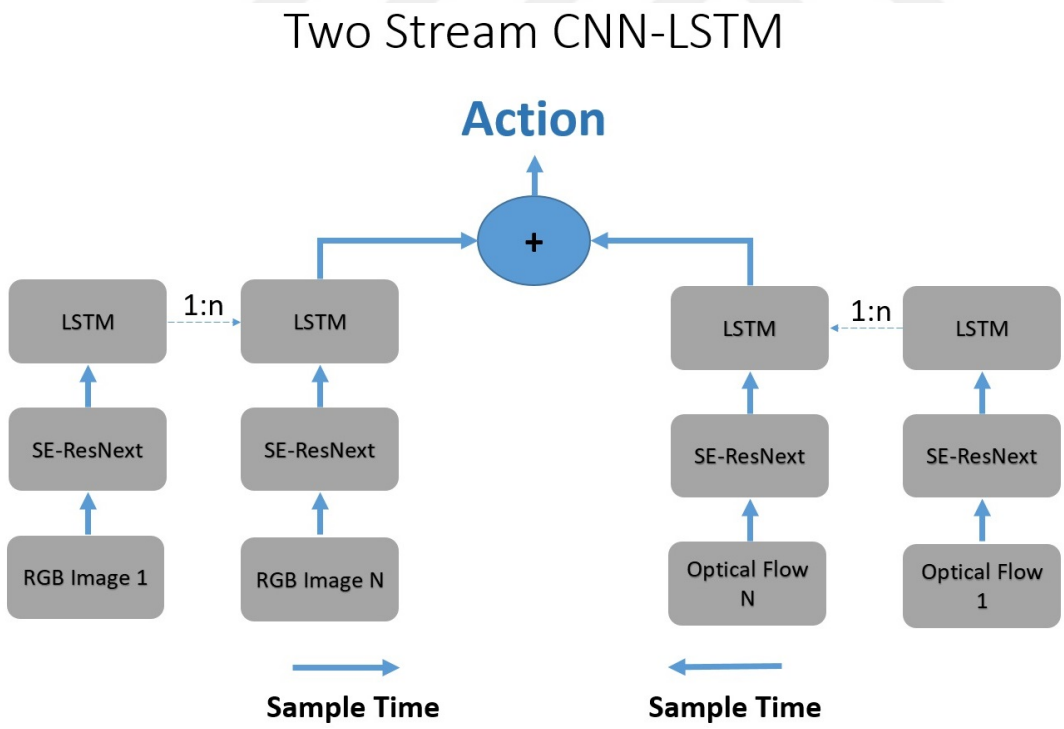


Figure 2.39: Proposed Method.

CHAPTER 3

EXPERIMENTAL RESULTS

In this chapter, first we introduce the dataset used in the models. Then we introduce trained networks and their performances. Lastly, we compare the results. We trained twelve model in total belonging to the two different categories that are introduced in the literature survey section. We have developed a base model for both groups, which is known to be successful and compared this model with the state of the art version. Thereby, we had an opportunity to discuss inner-class and inter-class performances. All models are trained with NVIDIA GeForce GTX 1060 GPU and Intel i5-6400 CPU.

In the first category, we worked on LSTMs. As introduced in section 2.5.2, LSTM could be used to process sequential data. To train the model with LSTM, features should be extracted to describe the given input image. In the literature survey section, we have investigated the evolution of convolutional neural networks. We have introduced VGGNet [37], Inception (GoogleNet [4], [2], [3], [44]), ResNet families (ResNet [39] and ResNext [5]) and depth-wise convolution based models (MobileNet [41], [6] and Xception [15]). These CNNs have been widely used as features extractors for the last few years and we have tested their performance in our CNN-LSTM models. For the CNN-LSTM networks we have used networks as feature extractor and trained eight different models. For the ResNext architecture, performance of block attention was also tested. Tested scenarios are listed below;

- Inception-LSTM
  - InceptionV3-LSTM (RGB)
  - InceptionResNetV2-LSTM (RGB)

- MobileNet-LSTM (RGB)
- Xception-LSTM (RGB)
- ResNext-LSTM
  - ResNext-LSTM (RGB)
  - SE-ResNext-LSTM (RGB)
  - SE-ResNext-LSTM (Flow)
  - SE-ResNext-LSTM (Two Stream)

For the CNN-LSTM models; InceptionV3, InceptionResNetV2, MobileNet, Xception and ResNext architectures were used as feature extractors. We didn't use VGGNet due to memory limitations. For all CNN models, feature lengths and number of parameters are given in 3.6.

For the CNN-LSTM hybrids, InceptionV3-LSTM model was chosen as base model, since InceptionV3 has been widely used as feature extractor over the years. After obtaining good results with the base model, we have trained different models.

First, we have trained ResNext-LSTM model with RGB images. Then, as introduce in section 2.4, we have used squeeze and excitation (SE) blocks in ResNext. Compared to first model, it gave better results, which indicates SE attention produce more general features. After observing that, SE attention was used for feature extraction process. Next, we have changed the input and trained with optical flow images. Finally, we have created a two stream model, using these (RGB and Flow) models. To calculate two stream output, we have averaged their softmax scores in the end.

We have also trained InceptionResNetV2, MobileNet, Xception networks. Overall, 5 different architectures were used for feature extraction process.

For the 3D-CNN models, we have implemented C3D model [20] and I3D model [21]. C3D is one of the milestone for 3D-CNNs. Inspired by VGGNet, C3D uses only 3x3x3 convolutions. This is a straightforward architecture and it is used as base network for our 3D-CNN models. On the other hand, second network, I3D, is state of the art for 3D-CNNs. Also it has very different architecture. I3D network is created

by inflating the Inception-V1 networks into 3D. This gives them the opportunity to use pre-trained ImageNet weights by that way. For the 3D-CNN networks we used two different (C3D and I3D models were used.) network and trained four different models. For the I3D different inputs and two stream performance was tested.

Tested scenarios are listed below;

- C3D (RGB)
- I3D
  - I3D (RGB)
  - I3D (Flow)
  - I3D (Two Stream)

3.1 The Dataset

ChaLearn Looking at People Dataset: This dataset has been obtained for "Isolated Gesture Recognition" challenge and released at International Association for Pattern Recognition 2016 (ICPR '16). Database includes 249 gesture videos. Each RGB video represents one gesture only, and clips are performed by 21 different individuals. Dataset given as *.avi files. In order to train our algorithms we have extracted image sequences for each video.

Since the original dataset contains too much data, it is hard to train with limited resources. To compare our models, we have decided to create sub-dataset. So, we have created 40 class dataset from the original dataset. There are sub-groups in original dataset, we tried to sample classes from each sub-groups. 40-class dataset is given in appendix (6.1, 6.2). We shuffled the videos in this dataset and created training, testing and validation datasets. We used 60% of videos for training, 20% validation and 20% for testing. 5972 samples are used for training and 1493 samples are used for validation. Number of samples for each classes are given in given in appendix (6.3, 6.4). The videos in the dataset are on average 3 seconds long. To represent each gesture, 40 frames are extracted and optical flow images are also calculated to feed the algorithms when needed.

Example from the dataset is given in 3.1. In the given example, a woman raises her hand up and lowers it back. RGB samples are given on the left and corresponding optical flow images are given in the right. We can see that, in the beginning, when the scene is constant, there aren't any change in optical flow images. But when the action starts, displacement vectors are calculated and based on the direction and magnitude of the displacement vector, images are created.

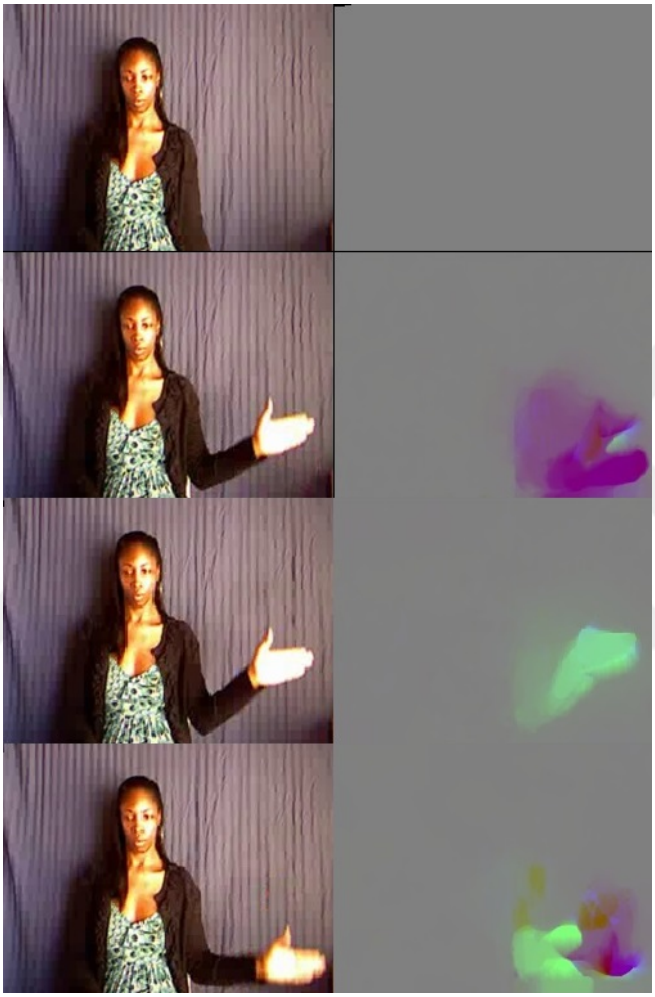


Figure 3.1: Dataset Example (RGB images are given on the left and optical flow images are given on the right).

3.2 The Networks

In this subsection we introduce implementation of each networks in details. As mentioned above, we created twelve different setup in two category. We wanted to com-

pare networks with each other. We also proposed our network which achieves better performance compared to other networks which have nearly the same number of parameters and also achieved nearly equal results with some state of the art networks.

For all models, we trained our models for 250 epochs or until validation accuracy did not improve over 25 successive iterations.

Training dataset was used to train model and validation dataset was used to calculate validation accuracy after each epoch. Final model scores were calculated using test dataset.

For the hyper parameter search, we have tuned parameters for different models. For the CNN-LSTM networks, learning rate and momentum values are fixed for Inception-LSTM architecture and these parameters are used for all CNN-LSTM models. Same procedure was fallowed for 3D-CNNs. Also, number of input frames for the model is investigated. We have tried 10-25-40 frame inputs for a test model and observed that increasing the number of input frame increases the performance.

3.2.1 CNN-LSTM Networks

For the CNN-LSTM Networks we have used CNN as feature extractor. We have extracted features before the final layers. Extracted features are then fed to the LSTM modules. Here we have used LSTM as many to one configuration. Each extracted features are given as input and in the end we take classification result as the output of the LSTM. For this cascaded configuration, we did not train the CNN weights due to memory limitations. We can say that if it was trained as end to end configuration where CNN weights are also updated with respect to error, it would give better results. But still extracted features are powerful enough to use directly.

As stated above we have created two different network. Each will be introduced in the subsections. Then we will compare their results.

3.2.1.1 Inception-LSTM Network

For the tandem Inception-LSTM architectures, we used InceptionV3 and Inception-ResNetV2 architectures and extracted features at the final pooling layer of the networks. Extracted feature lengths are 1x2048 and 1x1536 respectively. We used pre-trained ImageNet weights. For the tandem approach weights are not trained, so we extracted features before starting the training. After extracting the features, we created a simple LSTM network, then passed the features from our CNN to this model.

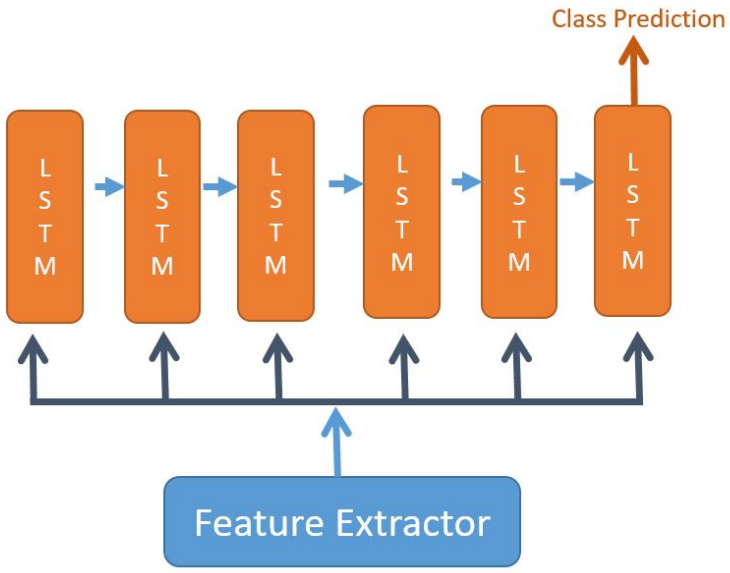


Figure 3.2: Inception-LSTM Network.

For the LSTM part, inputs are given as seqLength x featuresLength. We used 2048 LSTM unit with 0.5 drop out probability, followed by fully connected layer with 512 node with 0.5 drop out probability. Finally, a fully connected layer with number of output class is added with softmax activation to get the output probabilities. We used categorical cross entropy loss and Adam optimizer with initial learning rate $10^{-}5$ and learning decay $10^{-}6$.

In Table 3.1, number of trainable parameters for the InceptionV3-LSTM network are shown for each layers. Dense layer represents fully connected networks, which are 512 and 40 (number of class) respectively. Here we only give the trainable part, since Inception weights are fixed.

Table 3.1: Trainable parameters in Inception-LSTM Network.

Layer (type)	Number of Parameters
LSTM	33562624
Dense 1	1049088
Dropout	0
Dense 2	15390

Total:	58,478,886
Trainable:	34,627,102
Non-trainable (Inception-V3 part):	23,851,784

We used 5972 samples for training and 1493 samples for validation. We used 5 batch for each iteration and trained 250 epoch. For the Inception-V3 architecture, we can see that validation accuracy oscillates around 75%, for the InceptionResnet-V2 validation accuracy reaches 84.5%. Accuracy and loss plots are given in 3.3 and 3.4, respectively.

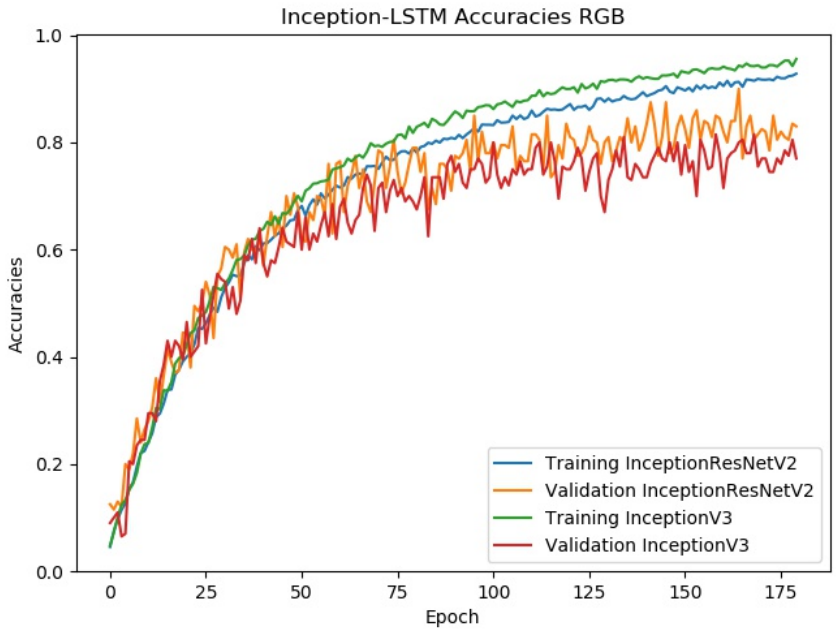


Figure 3.3: Accuracy over epochs for Inception-LSTM Network.

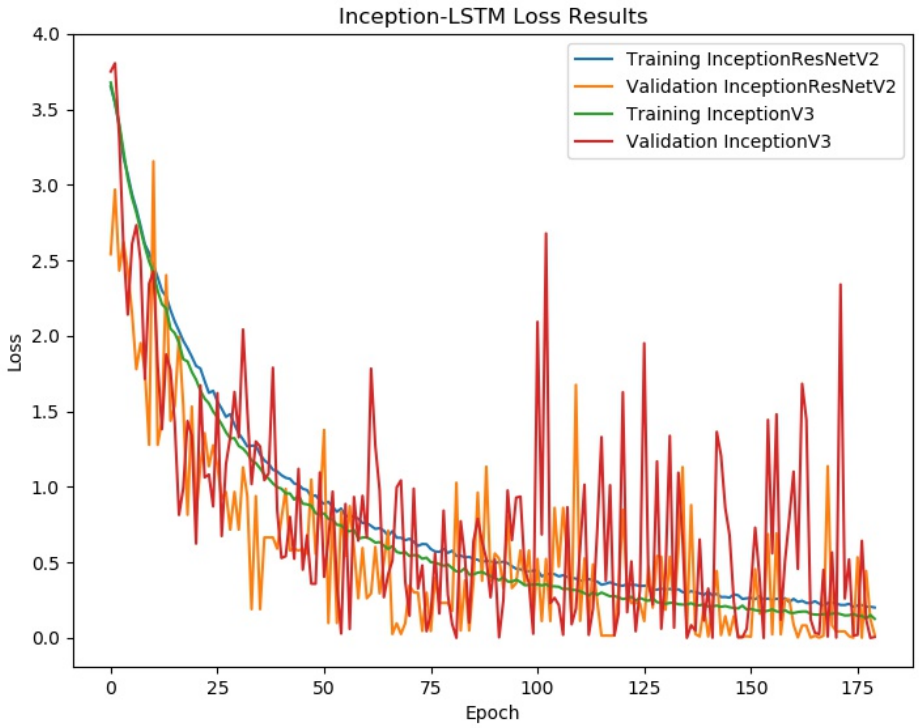


Figure 3.4: Loss over epochs for Inception-LSTM Network.

3.2.1.2 MobileNet-LSTM and Xception-LSTM Networks

For the MobileNet-LSTM and Xception-LSTM architectures, again, features are extracted at the final pooling layer of the networks. Extracted feature lengths are 1x1024 and 1x2048 respectively.

For these networks, same LSTM architecture (2048 LSTM unit with 0.5 drop out probability) and same training procedure (categorical cross entropy loss and Adam optimizer) was followed. For the MobileNet architecture, we can see that validation accuracy reaches 82.9%, and for the Xception validation accuracy reaches 86.5%.

Accuracy and loss plots are given in 3.5 and 3.6, respectively.

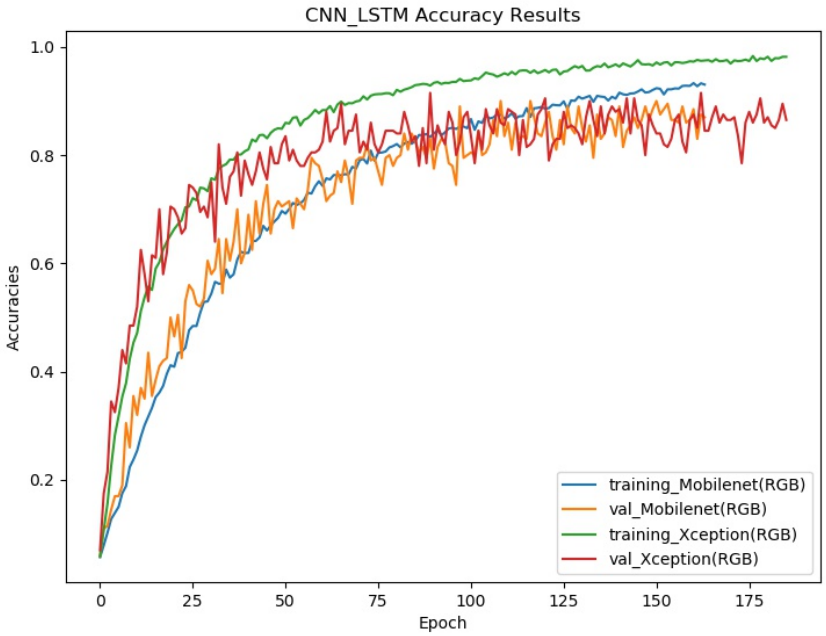


Figure 3.5: Accuracy over epochs for MobileNet-LSTM and Xception-LSTM Networks.

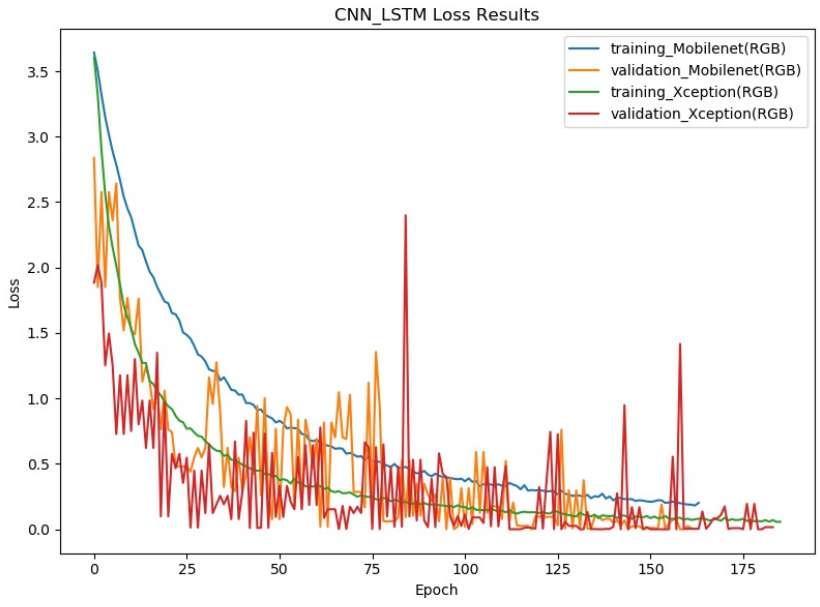


Figure 3.6: Loss over epochs for MobileNet-LSTM and Xception-LSTM Networks.

3.2.1.3 ResNext-LSTM Network

We have trained four different ResNext-LSTM networks. In the final configuration, we have combined two SE-ResNext models. First, we have trained ResNext-LSTM network with RGB images. Then, to investigate the effect of SE block in the model, we have added SE blocks for ResNext. After seeing that, SE blocks increase the accuracy, we have also trained SE-ResNext-LSTM network for flow images. Finally, RGB and Flow networks with SE blocks are used in two stream configuration.

In figure 3.7, final version of proposed method is depicted. Here, both networks use SE-ResNext-LSTM architecture. First network is trained (on the left) with RGB images and second (on the right) is trained with flow images. We should emphasise that, networks are not trained together, to calculate the output, we can simply calculate each network output and take the average of both network results.

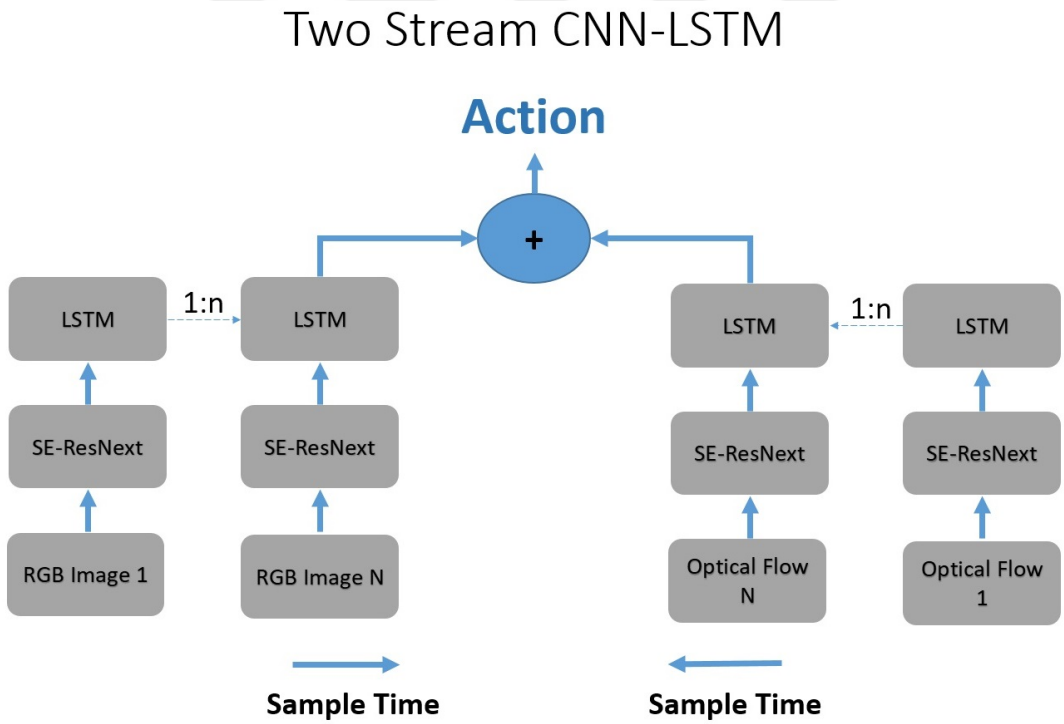


Figure 3.7: Two Stream SE-ResNext50-LSTM Module.

For the proposed architecture we have extracted features at the top layer of ResNext-50 network. We have used pre-trained ImageNet weights. Again, like for Inception V3-LSTM approach, weights are not trained for CNN. After extracting the features

we build a similar LSTM network, then pass the features from our CNN to this model.

For all three configuration, we have used 32 LSTM unit with 0.5 drop out probability, followed by fully connected layer with 512 node with 0.5 drop out probability. Finally, a fully connected layer with number of output class is added with softmax activation to get the output probabilities. Like for the other architectures again, we have used categorical cross entropy loss and Adam optimizer with initial learning rate $10^{-}5$ and learning decay $10^{-}6$.

Table 3.2: Trainable parameters in SEResNext-LSTM Network.

Layer (type)	**Number of Parameters**
SE-ResNext50	25579120
LSTM	26218624
Dense 1	16896
Dropout	0
Dense 2	20520

Total:	51,835,160
Trainable:	26,256,040
Non-trainable (SE-ResNext50 part):	25,579,120

From table 3.2, we can see that this model actually uses less parameters then the first model. Non-trainable parameters represent SE-ResNext50 modules, ResNext50 model has similar number of parameters. Dense layer represents fully connected networks, which are 512 and 40(number of class) respectively.

First we trained ResNext-LSTM network for RGB images. After 40 epoch training, we have obtained 98% training and 80% validation accuracy.

Then we added SE block for ResNext, after 40 epoch training, we have obtained 97% training and 86% validation accuracy for SE-ResNext-LSTM network. This is a good indicator that, using SE block creates more robust features.

Third, we trained SE-ResNext-LSTM network for flow images and obtained 97% training and 72% validation accuracy. It is clear that RGB inputs gives better results,

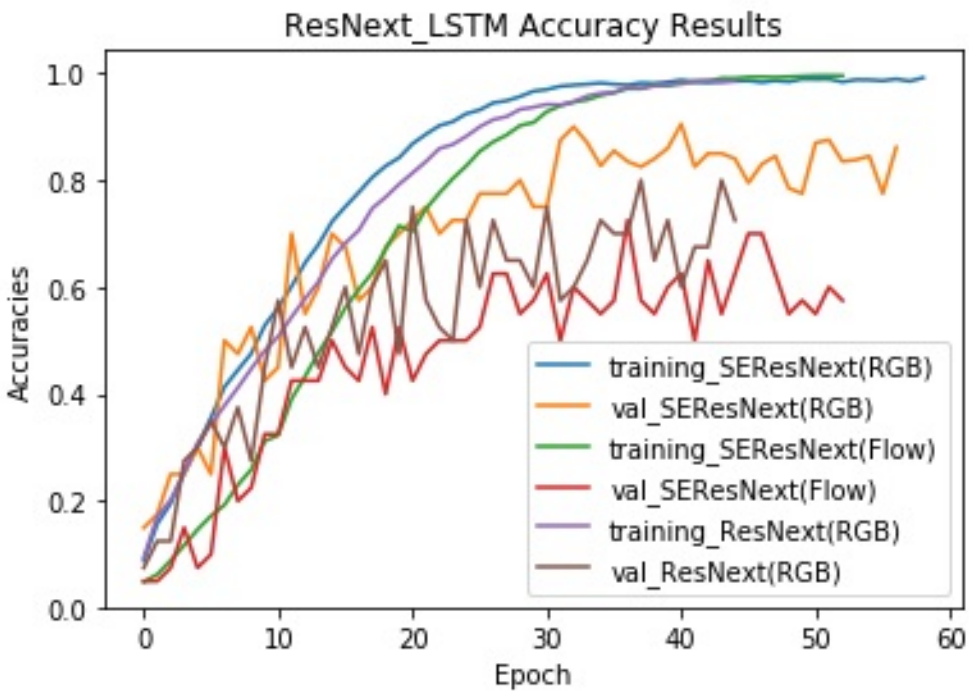


Figure 3.8: Accuracy over epochs for ResNext-LSTM Networks.

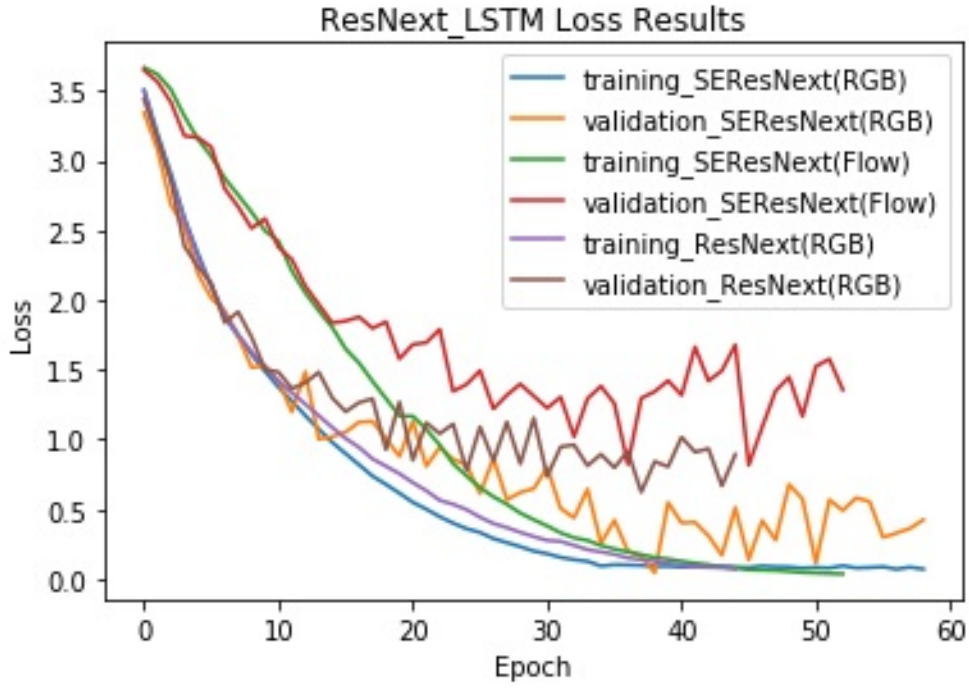


Figure 3.9: Loss over epochs for ResNext-LSTM Networks.

compared to flow images. Accuracy and loss graphics for all models are given in figure 3.8 and 3.9.

Finally, we have combined SE-ResNext models (RGB and Flow) in two stream configuration. From 3.10 to 3.12, we shown the each model scores and combined model performance. Also confusion matrix of each models are given. To calculate two stream score, we have averaged their softmax scores in the end. Here we can conclude that; although flow model could not reach high accuracy rates by itself, when it is used with RGB model, it can still increase the overall accuracy. Detailed analysis are given in 3.9.

For an N class classification problem, confusion matrix is an N×N matrix, where each row showing the true class (from test set) and each column showing the predicted class for the given class. For a given class (i) in a row, each column elements represent the number of items that were classified as being in class (j), where i=1:N and j=1:N. Diagonal line values (i=j) give the truly classified numbers for each class (true positives, TP). Precision, recall and F1 scores can also be obtained after calculating true negative (TN), false positive (FP) and false negative (FN) values.

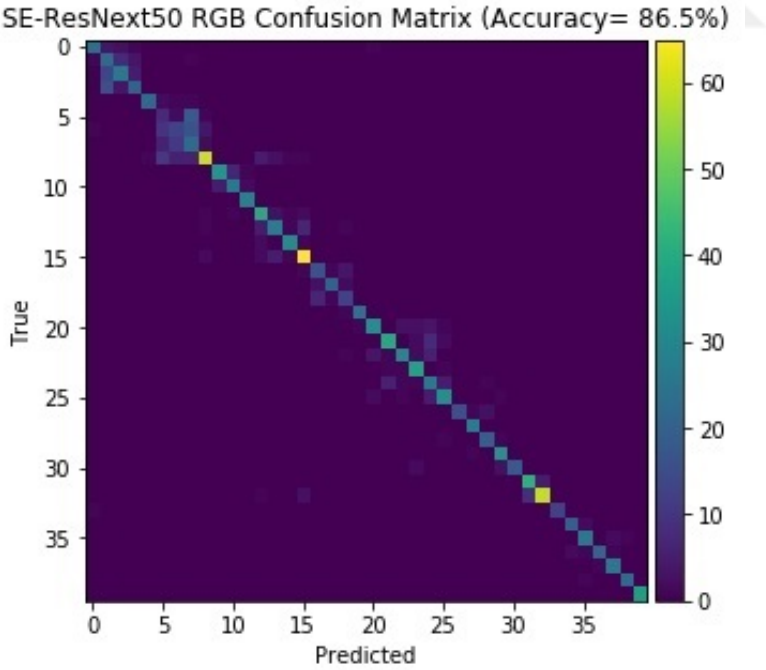


Figure 3.10: SE-ResNext Results and Confusion Matrices (RGB).

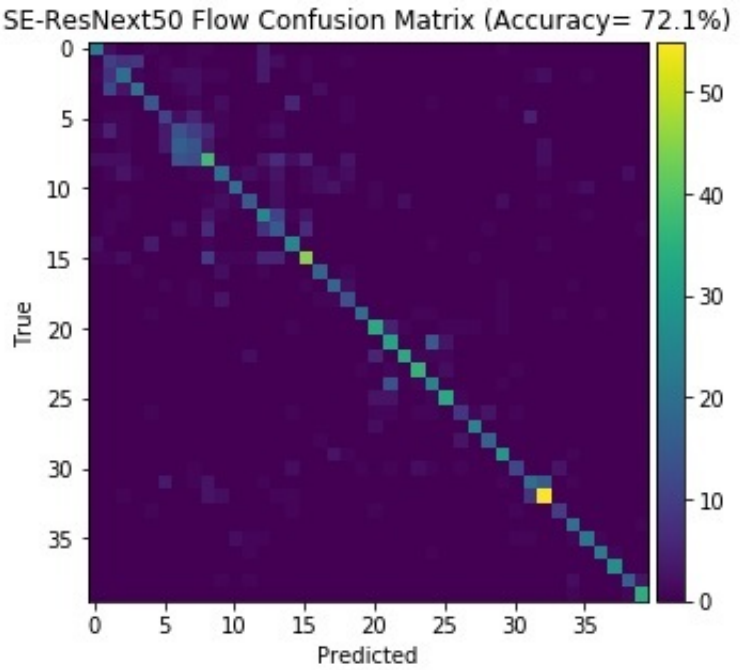


Figure 3.11: SE-ResNext Results and Confusion Matrices (Flow).

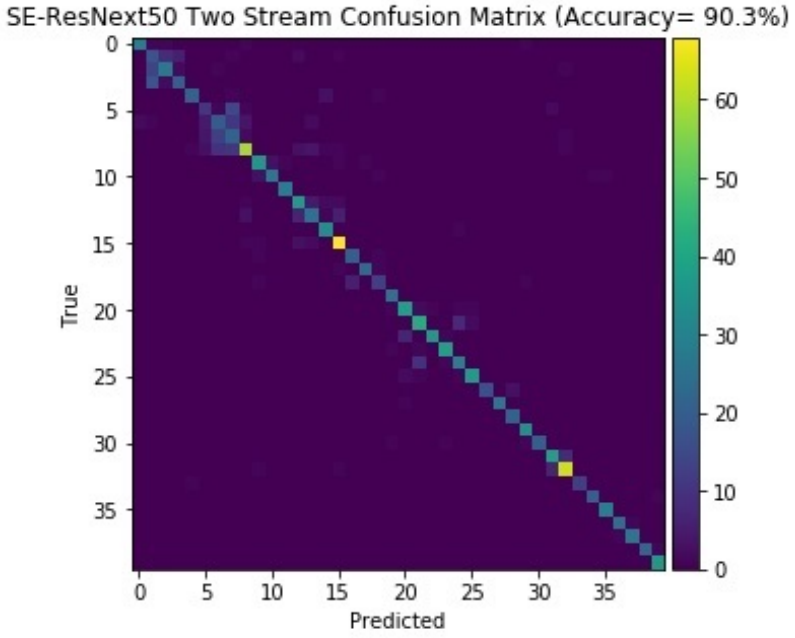


Figure 3.12: SE-ResNext Results and Confusion Matrices (Two Stream).

3.2.2 3D-CNN Networks

For the 3D CNN models; we have trained two different model. For the first model we have used [20] known as C3D and for the second model we have used [21] known as I3D.

3.2.2.1 C3D Model

For 3D-CNN model we have create a model similar to [20]. Extracted image sequence are feed to the model, images are resized to (80x80x3).

3x3x3 kernel are used to train the model. When the number of filter is changed, 1x2x2 pooling with 1x2x2 stride is applied. 3-D convolution with filter sizes 32-64-128-128-256-256 are applied sequentially. After these layers two fully connected layer with 1024 neurons followed by drop out layer with 0.5 drop out probability are added. Finally, a fully connected layer with number of output class is added with softmax activation to get the output probabilities. Proposed architecture is given in figure 3.13;

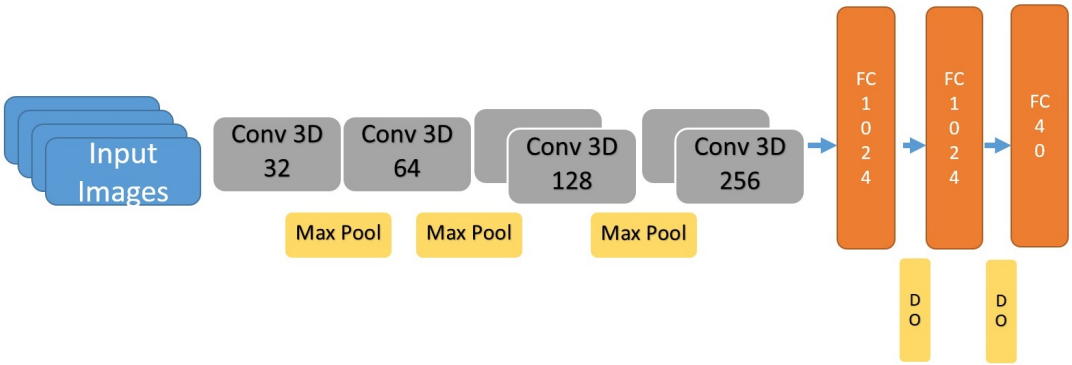


Figure 3.13: C3D Network.

For this model, we have trained the model from scratch, since there are not any pre-trained weights for our model. Model details and number of parameters for each layer is given in table 3.3. We have trained the network for 50 iteration, accuracy and loss graphics for this models are given in figure 3.14 and 3.15. Again, like Inception-LSTM model, accuracy remains around 70%.

In Table 3.3, we can see that, unlike the hybrid models, all parameters are trainable. Here, there is no need an external feature descriptor, since 3D-CNNs can learn both spatial and temporal features. In Table 3.3, Conv3D layers represent convolution layers, pooling layers are not shown. Dense layers represent fully connected layers.

Table 3.3: Trainable parameters in C3D Network.

Layer (type)	**Number of Parameters**
Conv3D 1	2624
Conv3D 2	55360
Conv3D 3	221312
Conv3D 4	442496
Conv3D 5	262400
Conv3D 6	524544
Dense 1	31458304
Dropout	0
Dense 2	1049600
Dropout	0
Dense 3	30750

Total:	34,047,390
Trainable:	34,047,390
Non-trainable:	0

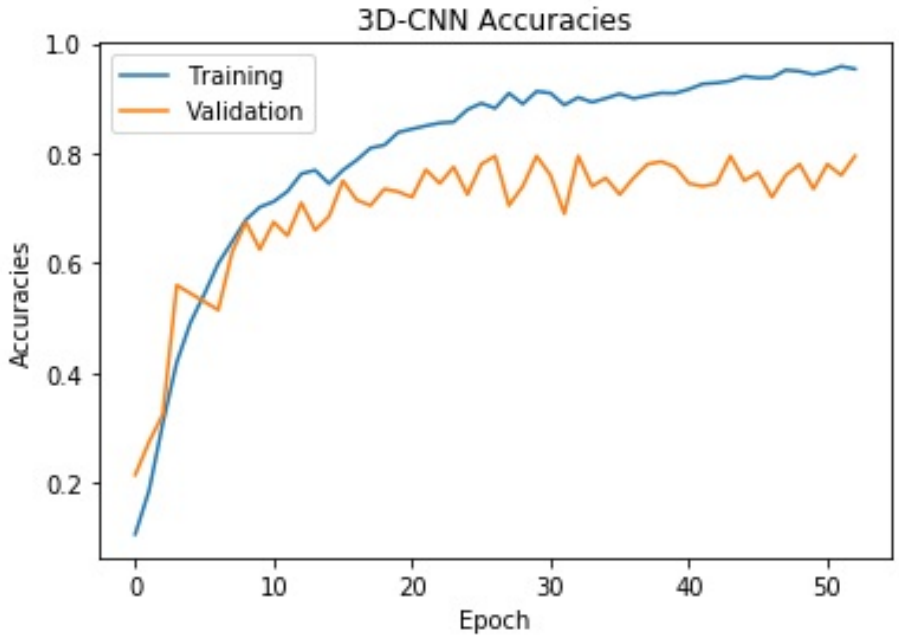


Figure 3.14: Accuracy over epochs for C3D Network.

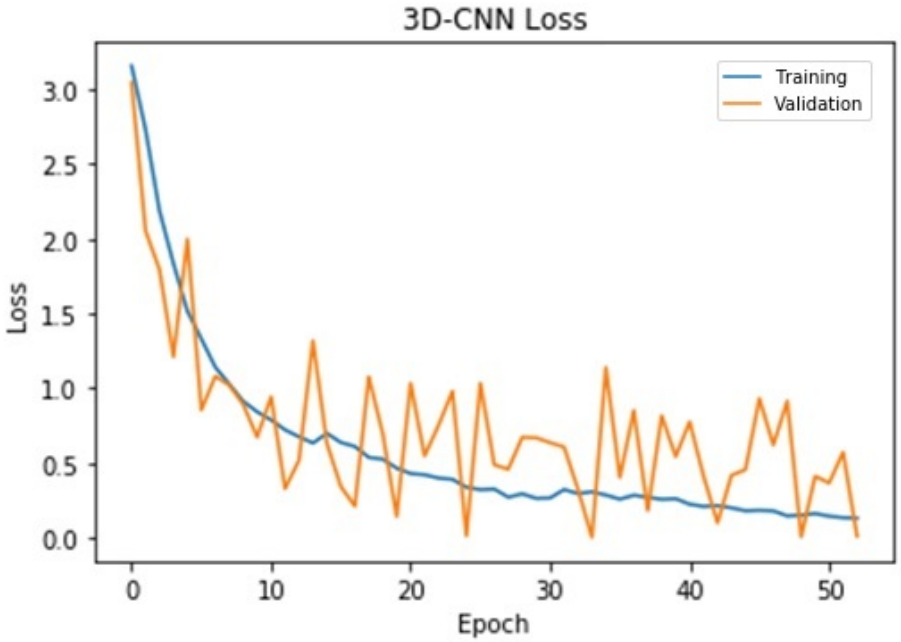


Figure 3.15: Loss over epochs for C3D Network.

3.2.2.2 I3D Model

For I3D model we have used the model provided by [21]. As discussed in the literature section, they have inflated the Inception-V1 networks into 3D and used pretrained ImageNet weights by that way. Then they have trained the networks on Kinetics dataset [45], which is also introduced by the same group. Kinetic is a large human actions dataset that contains 240k training videos total. Dataset has 400 different class and each class has at least 400 samples. They have published both the model and weights for Kinetic dataset for RGB and flow images.

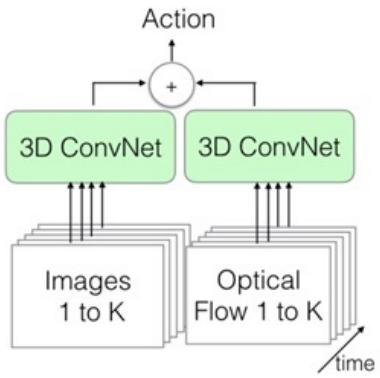


Figure 3.16: I3D Network [21].

Models were initialized with Kinetic weights. Unlike from C3D, ability to use pretrained weights give us the opportunity to initialize out networks with learnt parameters. Then we can tune these parameters according to our new dataset, which is known as transfer learning.

We have replaced the top layer with our 40 class module and first fine-tuned only the top layer for 3 epoch then trained entire network for 20 epochs. We have used this strategy for both RGB and flow images. In the end, to get the final decision, model softmax scores are averaged. For both RGB and flow networks accuracy and loss graphics are given in figure 3.17 and 3.18.

As we can see, I3D models achieve best results, while fitting quickly. This is because, Inception weights, trained on Kinetics dataset, are powerful enough to describe the each scene, few iteration is sufficient to adjust the weights.

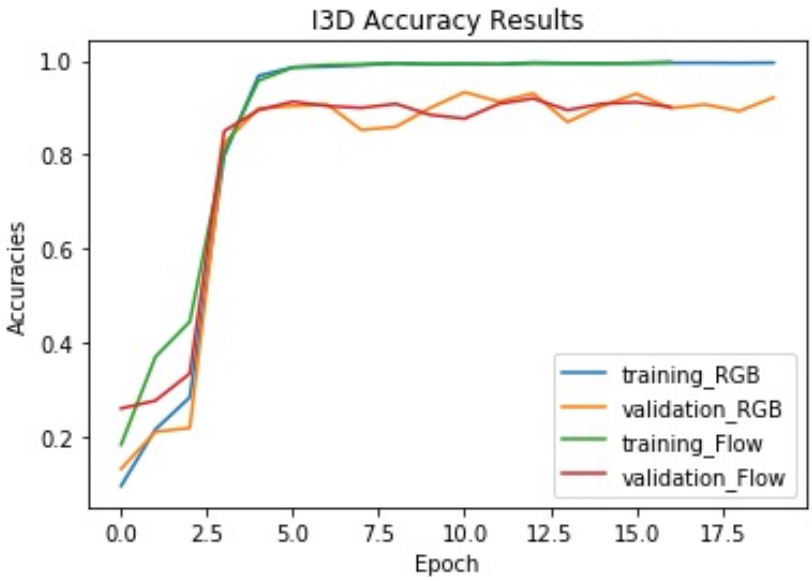


Figure 3.17: Accuracy over epochs for I3D Networks.

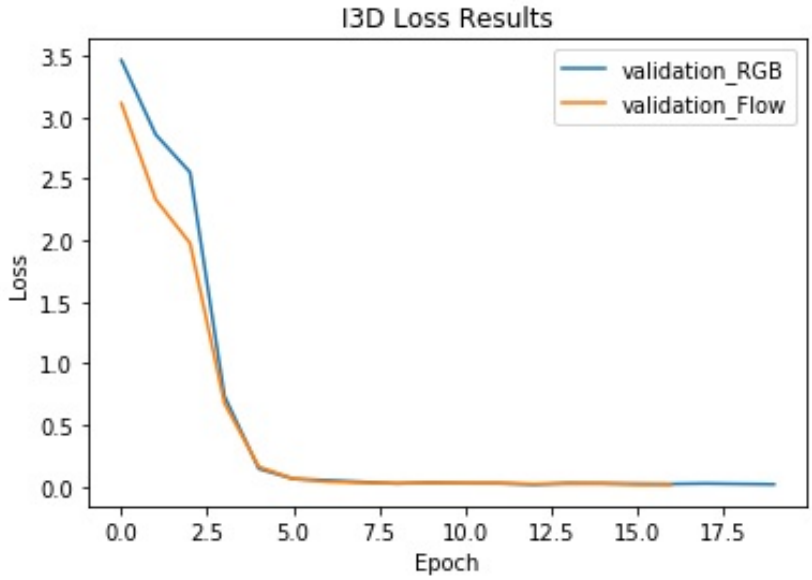


Figure 3.18: Loss over epochs for I3D Networks.

From 3.19 to 3.21, we shown the each model scores and combined model performance. As it is stated, we have averaged their softmax scores in the end. Again confusion matrices are also given. Again, detailed analysis are given in 3.8.

Number of trainable parameters are also shown in table 3.4. While the proposed architecture, introduced in the last section, contains about 26 million parameters for

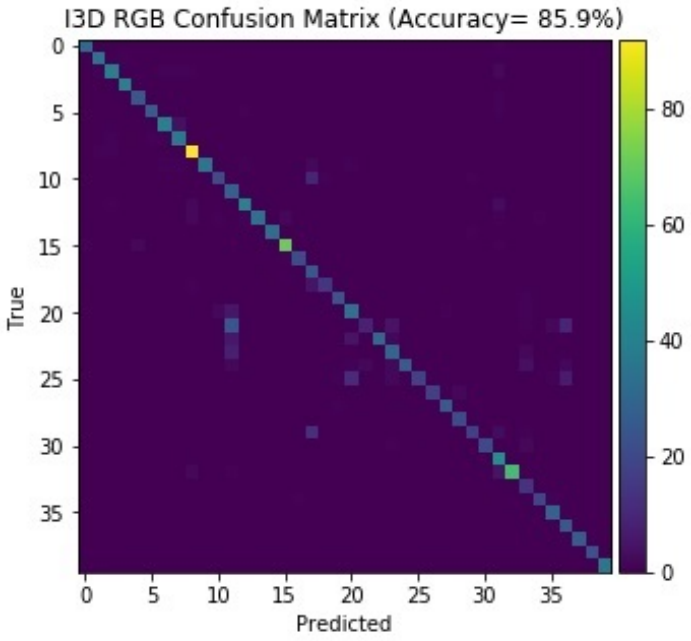


Figure 3.19: I3D Results and Confusion Matrices (RGB).

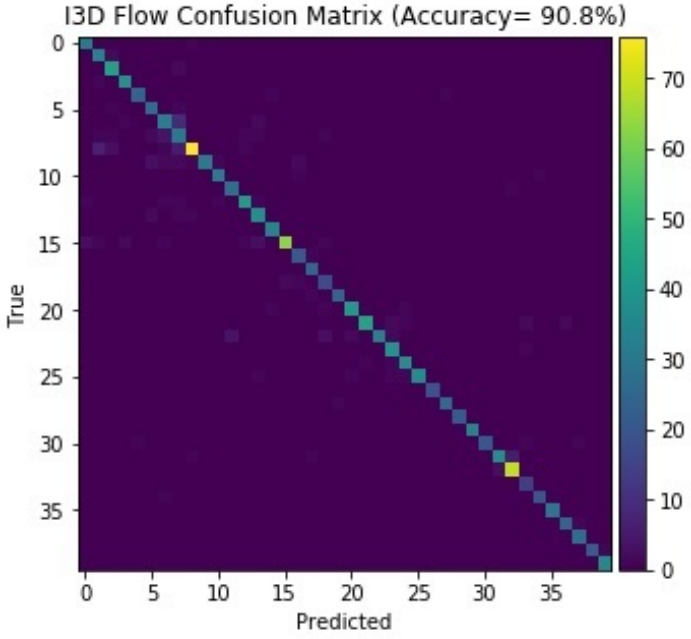


Figure 3.20: I3D Results and Confusion Matrices (Flow).

each model, here each model contains nearly half of it but still achieves equally good results. The reason might be the Kinetics dataset, since it contains lots of train-

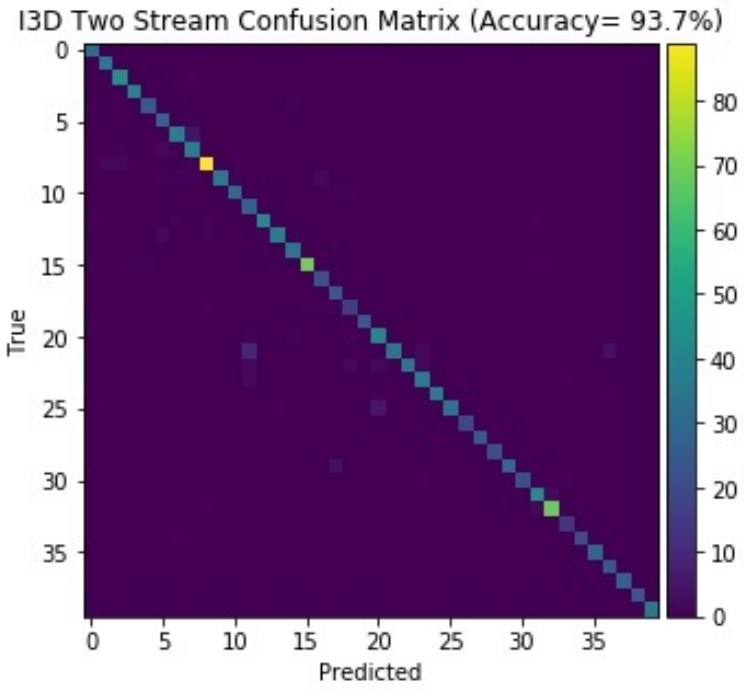


Figure 3.21: I3D Results and Confusion Matrices (Two Stream).

ing samples. These models contains roughly three times the parameter of original Inception-V1 architecture, since they are the inflated version of it.

Table 3.4: Trainable parameters in I3D Network.

	I3D_RGB	**I3D_Flow**
Total:	12,335,544	12,313,592
Trainable:	12,320,984	12,299,032
Non-trainable:	14,560	14,560

3.3 Results and Analysis

All the model results are given in the related subsections. Finally, in this chapter we will compare the results and comment on.

Number of Parameters vs Accuracy Comparison : In table 3.5, we have showed each model parameters and accuracy results. Here we can say that I3D models achieve best performance with minimum parameters. Our model also achieves similar accuracy but it uses more parameters compared to I3D model. When we used I3D models in two stream configuration, it has nearly the same number of parameters with our single model but achieves the highest performance. We can see that using two stream configuration improve both models (SEResNeXt50 and I3D) performance. When we compared I3D and SEResNeXt50 models; we can say that, RGB networks gives nearly equal performance, but flow network for I3D model gives much better result compared to SEResNeXt50 counterpart. This difference effects the overall two stream results. In our tests, flow images gave better results when it is used in 3D-CNNs compared to CNN-LSTM networks.

Table 3.5: Number of Parameters vs Accuracy Comparison.

Model	**Trainable (#)**	**Accuracy**
InceptionV3-LSTM	34,627,102	0.745
MobilenetV2-LSTM	26,243,624	0.829
InceptionResnetV2-LSTM	30,437,928	0.846
Xception-LSTM	34,632,232	0.865
ResNeXt50-LSTM(RGB)	26,256,040	0.801
SE-ResNeXt50-LSTM (RGB)	26,256,040	0.865
SE-ResNeXt50-LSTM (Flow)	26,256,040	0.721
SE-ResNeXt50-LSTM (Two Stream)	52.512.080	**0.902**
C3D	34,047,390	0.795
I3D (RGB)	12,320,984	0.859
I3D (Flow)	12,299,032	0.908
I3D (Two Stream)	24,620,016	**0.937**
SE-ResNeXt50-LSTM & I3D (Two Stream)	38,555,072	**0.943**

Effect of Feature Extractor Networks: When we compare the CNN-LSTM networks, we can say that feature descriptors are very important. Although weights are not trained for both CNN models, impact of choosing the better model is quite sig-

nificant. Actually, both Inception-V3 and ResNext50 models contains nearly same number of parameters as it can be seen in table 3.6. This is a strong indicator that ResNext50 has a better representational power compared to Inception-V3 while maintaining computational cost roughly the same.

Table 3.6: Number of Parameters in CNN Networks.

CNN Model	Number of Parameters
MobileNetV2	3,538,984
Inception-V1	6,797,700
Xception	22,910,480
Inception-V3	23,851,784
ResNeXt50	25,097,128
SE-ResNeXt50	25,579,120
ResNet50	25,636,712
InceptionResnetV2	55,873,736
VGG-16	138,357,544

Another important outcome is, squeeze and excitation blocks improve the performance. Compared to the base ResNext model, SE-ResNext model improved the accuracy about 7% for RGB models. So, when CNN models used as feature extractor SE block produce more general features, which helps accuracy increase. We can conclude that, SE blocks learn global information to select informative features and suppress less informative ones. On the other hand, computational burden of SE blocks is quite low. We can see from table 3.6, number of parameter difference is nearly 500k. It can be seen that ResNet and ResNext architectures are also have nearly same number of parameters, as it is pointed in the literature section.

Effect of Pre-Trained Weights : Using pre-trained weights are also crucial for any network. We can see that, while C3D model struggles to achieve good accuracy, I3D model fits quickly. Although the only reason is not the pre-training, since networks are also different, it is clearly one of the biggest reason.

Class-wise Analysis : In this subsection, we will be investigating the class-wise accuracies. Classes that are achieving poor performance will be analysed. These analyses will be given for I3D and SE-ResNext-LSTM architectures and for all three configurations (RGB, optical flow and two stream). For convenience, investigated classes are listed in table 3.7.

Table 3.7: Mostly Confused Classes.

Class Index	**Class Group**	**Class Name**
1/C-027	Italian Gestures	NonMiFrega
2/C-030	Italian Gestures	VieniQua
3/C-049	Gestuno Disaster	thunderstorm_orage
5/C-065	Gestuno Landscape	volcano_volcan
6/C-071	Gestuno Small Animals	butterfly_papillon
7/C-072	Gestuno Small Animals	cat_chat
8/C-086	Gestuno Topography	harbour_port
10/C-001	Mudra1	Ardhachandra
17/C-015	Mudra2	Mrigashirsha
18/C-018	Mudra2	Tamrachuda
20/C-033	Chinese Numbers	liu
21/C-035	Chinese Numbers	shi
22/C-037	Chinese Numbers	wu
23/C-039	Gestuno Colours	colour_couleur
25/C-043	Gestuno Colours	yellow_jaune
36/C-136	Diving Signals2	CannotOpenReserve

Mostly confused classes for I3D and SE-ResNext models are given in table 3.8 and 3.9, respectively. When we look at the table 3.8 and 3.9, we can see that confusion occurs between classes that share the same class groups. This is an expected result, since for a given class, it is more likely to share some common features with its group members. Also, for a given network architecture, we see that same classes are confused for all three configuration. This is an indicator that, network architecture has more effect, compared to input characteristics.

For the RGB networks, 6 results are given for SE-ResNext model and 4 results are given for I3D model. Both model achieved nearly same results, as it is given in table 3.5. On the other hand, classes in which they fail are very different. Again, this is also true for other configurations.

Table 3.8: Class-wise analysis for I3D Results.

I3D

Two Stream

Class	Accuracy	Mostly Confused
c21	.69	c11
c22	.82	c11, c18
c25	.83	c20

Optical Flow

Class	Accuracy	Mostly Confused
c22	.75	c11, c18

RGB

Class	Accuracy	Mostly Confused
c10	.64	c17
c18	.71	c17
c21	.17	c23, c36
c25	.45	c20

We have grouped the classes that are confused with each other for each networks. For the classes 1, 2 and 3 (C-027, C-030, C-049), SE-ResNext-LSTM architecture produced low accuracy results. When we analysed these classes, we see that in all three actions, performer uses his/her one hand and produce repeating patterns. In class 1 (C-027), performer raises a hand to his/her chin and bring it back down. Class 2(C-030) the gesture is similar to calling a person with hand. In class 3(C-049), performer make a lightning gesture with his/her hand.

Another group could be given as class 5-6-7-8. Again for this group, SE-ResNext-

LSTM architecture produced low accuracy results. In these actions, two hands come together in the middle and some of them contains repeating actions. In class 5 (C-065), hands are joined to create a triangle in the middle. In class 6 (C-071), hands are joined in the middle in a butterfly shape and perform flapping wing movement. In class 7(C-072), hands are joined in the middle and create a circular figure with 2 fingers. Finally in class 8(C-086), two hands join in the middle and one moves behind the other.

We can see that, repeating actions could be problematic for SE-ResNext-LSTM. Also for the given examples, each groups contains similar patterns that leads to confuse with each other.

For the I3D model, class 10, 17 and 18 produced low accuracy results. Common side of these actions is, one hand rises up but there isn't any repeating pattern. In class 10 (C-001), one open hand is raised in the air. For class 17 (C-015), one hand is raised in the air and make rock and roll hand sign. Finally for class 18 (C-018), one hand is raised in the air and make hook-shaped figure.

Last example could be given for class 20-25. Here again, I3D model produced low accuracy results. For these classes, similar to previous example, one hand rises up but there isn't any repeating pattern. Additionally, fingers are used in these actions.

For I3D networks, it is seemed repeating actions didn't create any problem. Conversely, constant movements produced lower scores compared to SE-ResNext-LSTM architecture.

Effect of Two Stream Configuration : In this subsection, we will investigate the effect of two stream configuration for both architectures. In tables 3.10 and 3.11, each class scores are given for RGB, optical flow and two stream configuration. Tables are colorized in three different colour; red, green and yellow. For the RGB and optical flow results, colours are chosen comparing their scores against each other. The one with higher accuracy is colorized with green and the other one is colorized with red. If their accuracies are equal, then both are colorized with yellow. For the two stream results, given cell is colorized with green only if using two stream configuration achieves better results. Yellow is used when two stream score could be obtain

Table 3.9: Class-wise analysis for SE-ResNext Results.

SE-ResNext

Two Stream

Class	Accuracy	Mostly Confused
c1	.59	c2, c3
c2	.69	c1
c3	.60	c1
c5	.5	c7
c6	.53	c7
c7	.61	c6
c8	.67	c6, c7

Optical Flow

Class	Accuracy	Mostly Confused
c1	.37	c2, c3
c2	.52	c1
c3	.60	c1
c5	.5	c7
c6	.4	c7
c7	.42	c6
c8	.44	c6, c7
c9	.55	c2, c13

RGB

Class	Accuracy	Mostly Confused
c1	.72	c2, c3
c2	.67	c1
c3	.61	c1
c5	.4	c7
c7	.64	c5, c6
c8	.68	c5, c6, c7

with one network (RGB or flow). Red is used when using two stream configuration decrease the overall accuracy.

For the SE-ResNext-LSTM architecture; given 40 classes, 15 classes achieve better results, 17 classes achieve equal results with one network and 8 classes give worst results when it is used in two stream configuration. Using two stream configuration increased the overall accuracy as it is discussed in accuracy comparisons.

For the I3D architecture; given 40 classes, 10 classes achieve better results, 21 classes achieve equal results with one network and 9 classes give worst results when it is used in two stream configuration. Again, using two stream configuration increased the overall accuracy.

Fusing Different Networks : In the previous chapter, effect of two stream configuration is discussed. While creating the two stream networks, we have used same architectures with different input characteristics. This is the general use case in the literature, as we have discussed we need capture both temporal and spatial features. On the other hand, different architectures produces different success rate for the static and repetitive gestures as we have pointed in the class-wise analyses section.

One alternative use case is using different architectures, while maintaining different input types. In Table 3.5 we can see that, both SE-ResNeXt50-LSTM (86.5%) and I3D (85.9%) produces nearly equal results for the RGB inputs. But as given in the class-wise analyses section, while CNN-LSTM networks produce better results for static inputs, I3D produce better results for repetitive inputs. We have decided to create hybrid two stream model. Since SE-ResNeXt50-LSTM gives better accuracy for the RGB inputs and I3D gives better accuracy for the flow inputs; we have used RGB network for the SE-ResNeXt50 and flow network for the I3D. Our expectation is, similar to performance increase obtained in different input types, the performance increase in static and repetitive videos. In that way, our network becomes more general which can capture temporal and spatial features. Obtained results are given in table 3.12, colorization is used in similar manner;

For this hybrid configuration; given 40 classes, 12 classes achieve better results, 23 classes achieve equal results with one network and 5 classes give worst results when

Table 3.10: Se-ResNext-LSTM Class Results.

Classes	RGB	Flow	All	Classes	RGB	Flow	All
0	0.84	0.81	1.00	**20**	0.82	0.84	0.95
1	0.72	0.37	0.59	**21**	0.84	0.71	0.88
2	0.67	0.52	0.69	**22**	0.77	0.84	0.91
3	0.60	0.60	0.60	**23**	1.00	0.97	1.00
4	1.00	0.70	0.96	**24**	0.75	0.70	0.82
5	0.40	0.50	0.50	**25**	0.88	0.90	1.00
6	0.37	0.40	0.53	**26**	1.00	0.65	1.00
7	0.64	0.42	0.61	**27**	1.00	1.00	1.00
8	0.68	0.44	0.67	**28**	1.00	1.00	1.00
9	0.90	0.55	0.95	**29**	1.00	0.94	1.00
10	0.91	0.70	0.94	**30**	0.91	0.62	1.00
11	1.00	0.68	1.00	**31**	0.95	0.48	0.89
12	0.91	0.62	0.87	**32**	0.86	0.80	0.91
13	0.74	0.46	0.69	**33**	1.00	0.92	1.00
14	1.00	0.85	1.00	**34**	1.00	1.00	1.00
15	0.92	0.65	0.96	**35**	1.00	0.83	1.00
16	0.91	0.95	1.00	**36**	0.96	1.00	1.00
17	0.96	0.88	1.00	**37**	1.00	1.00	1.00
18	0.80	0.85	0.80	**38**	1.00	0.83	1.00
19	1.00	0.92	1.00	**39**	1.00	1.00	1.00

Table 3.11: I3D Class Results.

Classes	RGB	Flow	All	Classes	RGB	Flow	All
0	0.96	0.96	1.00	**20**	0.77	0.93	0.93
1	0.97	0.94	1.00	**21**	0.17	0.86	0.69
2	0.86	0.95	0.97	**22**	0.71	0.75	0.82
3	1.00	0.97	1.00	**23**	0.75	0.97	0.92
4	0.96	0.92	0.96	**24**	0.68	0.95	0.95
5	0.93	0.93	0.96	**25**	0.45	0.90	0.83
6	0.91	0.73	0.88	**26**	0.90	1.00	1.00
7	0.92	0.73	0.92	**27**	0.96	0.96	0.96
8	0.97	0.80	0.94	**28**	1.00	1.00	1.00
9	0.86	0.74	0.86	**29**	0.51	1.00	0.88
10	0.64	0.94	0.94	**30**	0.87	0.87	0.91
11	1.00	0.96	1.00	**31**	1.00	0.82	0.93
12	0.87	0.89	0.91	**32**	0.88	0.95	0.95
13	0.81	0.90	0.90	**33**	1.00	1.00	1.00
14	0.97	1.00	1.00	**34**	0.90	0.95	0.95
15	0.94	0.84	0.93	**35**	1.00	1.00	1.00
16	0.95	0.95	1.00	**36**	1.00	0.96	1.00
17	1.00	1.00	1.00	**37**	1.00	1.00	1.00
18	0.71	0.85	0.80	**38**	1.00	0.95	1.00
19	0.96	0.96	1.00	**39**	1.00	1.00	1.00

it is used in two stream configuration. This two stream configuration increase the overall accuracy and achieves 94.3% accuracy rate. With this hybrid two stream architectures, we have obtained the highest score so far, which is a good indicator that using different architectures help accuracy increase.

Finally in table 3.13, all two stream results are listed together. Here we have used colorization in a different manner. Results are coloured red, yellow and green from the worst to the best.

Table 3.12: Hybrid Two Stream Class Results.

Classes	RGB	Flow	All	Classes	RGB	Flow	All
0	0.84	0.96	1.00	**20**	0.82	0.93	0.93
1	0.73	0.94	0.94	**21**	0.84	0.86	0.90
2	0.67	0.95	0.95	**22**	0.77	0.75	0.84
3	0.61	0.97	0.95	**23**	1.00	0.97	0.97
4	1.00	0.92	1.00	**24**	0.75	0.95	0.97
5	0.40	0.93	0.93	**25**	0.88	0.90	0.95
6	0.37	0.73	0.66	**26**	1.00	1.00	1.00
7	0.64	0.73	0.83	**27**	1.00	0.96	1.00
8	0.68	0.80	0.84	**28**	1.00	1.00	1.00
9	0.91	0.74	0.95	**29**	1.00	1.00	1.00
10	0.91	0.94	1.00	**30**	0.91	0.87	0.87
11	1.00	0.96	1.00	**31**	0.95	0.82	0.91
12	0.91	0.89	0.93	**32**	0.86	0.95	0.95
13	0.74	0.90	0.95	**33**	1.00	1.00	1.00
14	1.00	1.00	1.00	**34**	1.00	0.95	1.00
15	0.92	0.84	0.97	**35**	1.00	1.00	1.00
16	0.91	0.95	0.95	**36**	0.96	0.96	0.96
17	0.96	1.00	1.00	**37**	1.00	1.00	1.00
18	0.80	0.85	0.85	**38**	1.00	0.95	1.00
19	1.00	0.96	1.00	**39**	1.00	1.00	1.00

Table 3.13: All Two Stream Class Results.

Classes	ResNext	I3D	All	Classes	ResNext	I3D	All
0	1.00	1.00	1.00	**20**	0.95	0.93	0.93
1	0.59	1.00	0.94	**21**	0.88	0.69	0.90
2	0.69	0.97	0.95	**22**	0.91	0.82	0.84
3	0.60	1.00	0.95	**23**	1.00	0.92	0.97
4	0.96	0.96	1.00	**24**	0.82	0.95	0.97
5	0.50	0.96	0.93	**25**	1.00	0.83	0.95
6	0.53	0.88	0.66	**26**	1.00	1.00	1.00
7	0.61	0.92	0.83	**27**	1.00	0.96	1.00
8	0.67	0.94	0.84	**28**	1.00	1.00	1.00
9	0.95	0.86	0.95	**29**	1.00	0.88	1.00
10	0.94	0.94	1.00	**30**	1.00	0.91	0.87
11	1.00	1.00	1.00	**31**	0.89	0.93	0.91
12	0.87	0.91	0.93	**32**	0.91	0.95	0.95
13	0.69	0.90	0.95	**33**	1.00	1.00	1.00
14	1.00	1.00	1.00	**34**	1.00	0.95	1.00
15	0.96	0.93	0.97	**35**	1.00	1.00	1.00
16	1.00	1.00	0.95	**36**	1.00	1.00	0.96
17	1.00	1.00	1.00	**37**	1.00	1.00	1.00
18	0.80	0.80	0.85	**38**	1.00	1.00	1.00
19	1.00	1.00	1.00	**39**	1.00	1.00	1.00

CHAPTER 4

CONCLUSION

Gesture recognition is a very popular research area that has been studied over the years. Over time, gesture recognition has taken place in different problems such as surveillance systems, sign language recognition, robotic etc. and has been the target of machine learning researchers. In advance, countless solutions have been proposed to interpret the on-going actions. Researchers aimed to construct models to define the actions. Various models have been proposed. Some models try to use holistic representations, such representations are more likely to preserve the spatial and temporal structures of actions. Some other representations try to use local representations and aggregate the models. Still, these methods were hand crafted; therefore, these methods are not generic compared to deep learning based methods. After the outbreak of deep neural networks, researchers focused these models. Since then, deep learning based models have become widely-used for action recognition tasks.

Deep learning based models give opportunities to researchers to create generic models and train them in large scales datasets. Compared to hand crafted solutions, deep learning models could reach higher levels of accuracy, requires less amount of effort and easier to tune their parameters. Deep learning models could interpret hidden relations better than hand crafted models. Different kind of solutions also proposed for deep learning models. The most crucial challenge is how to deal with the temporal dimension. This is the main focus while creating the architectures. One solution is to create 3D-CNNS. 3D filters in these architectures can capture discriminative features along both spatial and temporal dimensions. Another method to capture the temporal information is to extract motion features. In these networks, pre-computed optical flow images are fed to networks. Different usage of optical flow images are possible,

but popular solution is to create two stream network, where one stream is trained with RGB images and the other is trained with flow images, and combine their results by late fusion. Last alternative is to create temporal sequence model. One of the most used networks for this task is the Recurrent Neural Network (RNN) that could take into consideration temporal data using recurrent connections in hidden layers. These temporal sequence models require informative features and CNNs have been widely used to extract features. Using CNNs as feature extractor, CNN-RNN (LSTM) hybrids could be created.

In this thesis, we aimed to compare the pros and cons of these different methods. To do so, we have created two main category; hybrid CNN-LSTM networks and 3D-CNNs. For both category, we have created a base model, which is proven to be successful and compare this model with state of art counterpart. Doing so, we had a chance to investigate inner-class and inter-class performances. Effect of using optical flow images and two stream configuration are also investigated for both category.

First, we have worked on CNN-LSTM networks. We have chosen five CNNs as feature extractors. Here purpose was to investigate the effect of feature extractor networks. In the literature section we have introduce different feature extractor networks and their contributions. VGGNet, Inception (InceptionV3 and InceptionResNetV2), ResNet families (ResNet and ResNext) and depth-wise convolution based models (MobileNet and Xception) have been discussed. VggNet architecture suggested that by using smaller kernels, we can increase the depth of the network without encountering the over fitting problem. InceptionV3 took that idea and also used network in network strategy and created mini networks in that way. They called these mini networks as Inception modules. Next, ResNet architecture introduced the residual connection to solve the over fitting problem. Finally, in ResNext architecture, we can see all the positive contributions until that point, ResNext uses residual function like ResNet to prevent degradation. Also uses sub-modules like Inception, calculate the response of each operation and concatenates the outputs at the end of the each module. Here they have introduced these sub-operations inside the given module as a dimension called “cardinality”. Convolution kernels inside the cardinality layers are chosen 3x3. Similar to ResNext, InceptionResNetV2 also takes advantage of Inception modules and residual connections.

We have trained eight different CNN-LSTM networks. When we compared the accuracies, InceptionV3-LSTM network achieves 74.5% and InceptionResNetV2-LSTM 84.6% accuracy. MobilenetV2-LSTM network achieves 82.9%, which is quiet significant since it contains less parameters compared the other feature extractor networks. Xception-LSTM network achieves 86.5% accuracy, one of the highest score for CNN-LSTM models.

ResNext-LSTM network achieves 80.1% accuracy. After observing that ResNext gives good results, we have continued our tests with ResNext model. First, we have tested an attention module called 'Squeeze and Excitation' block. SE blocks learn global information to select informative features and suppress less informative ones. We have obtained 86.5% accuracy for RGB inputs. Here, we can say that SE blocks help accuracy increase, since it produce more informative features. Finally, we have trained SE-ResNext module with optical flow images and created SE-ResNext-LSTM architecture. This architecture obtained 72.1% accuracy. It is clear that, RGB images for hybrid networks give better results compared to optical flow images. Finally, we have combined two separately trained SE-ResNext-LSTM networks (RGB and optical flow versions) and created a two stream architecture. This two stream version achieved 90.2% accuracy. To calculate two stream score, we have averaged their softmax scores in the end. Here we can conclude that; although flow model could not reach high accuracy rates by itself, when it is used with RGB model, it can still increase the overall accuracy.

Secondly, we have worked on 3D-CNNs. This time we have used C3D and I3D models. Our first model, C3D is one of the milestone for 3D-CNNs and it is used as a base model for our 3D-CNNs. C3D provides a straightforward network architecture. Using smaller size kernels make possible to create deep models without encountering over fitting problem. Compared to C3D model, I3D model has a very different design. One of the main drawback for 3D-CNN is inability to use ImageNet weights. I3D provides a solution for this problem by inflating the 2D-CNN architecture. They have used InceptionV1 architecture and inflated all weights into 3D. By that way, they have initialized the network with ImageNet weights, which provides great advantages.

In our C3D network, we have resized images to (80x80x3) to reduce the total number

of parameters. Even down sampling the input images, this network still contains about 34M parameters. Our model contains 6 layers before the fully connected layers. 32-64-128-128-256-256 filters are applied successively. When the number of filter is changed, 1x2x2 pooling with 1x2x2stride is applied. Kernel sizes are chosen as 3x3x3, as it is suggested in the literature section. For this model, we have used RGB images as inputs and obtained 79.5% accuracy. In our final model, we have trained I3D model. For this model we have used both RGB and optical flow images as inputs. RGB network achieved 85.9% accuracy and optical flow network achieved 90.8% accuracy respectively. When we combined these models in two stream configuration, we get 93.7% accuracy. Here optical flow model gives better result, compared to RGB model. We can conclude that optical flow images are more suitable for 3D-CNNs, compared to CNN-LSTM hybrids.

After evaluating all the tests for hybrid and 3D-CNN architectures, we can make the following inferences for our results;

- Using two stream configuration increased the accuracy for both configuration.
- RGB networks (SEResNeXt50-LSTM and I3D) showed nearly equal performances (86.5% and 85.9% respectively).
- Optical flow inputs for 3D-CNN architecture gave better performance compared to hybrid counterpart.
- Block attention module (SE) increased the accuracy since it creates more global features.
- CNN-LSTM networks produce better results for static inputs, 3D-CNNs produce better results for repetitive inputs.
- For two stream configurations, additional to different input types, using different architectures also help accuracy increase.

CHAPTER 5

FUTURE WORKS

The aim of this thesis is to investigate the different gesture recognition architectures and their advantages. We have evaluated two main category for gesture recognition problem namely, hybrid CNN-LSTM networks and 3D-CNNs. As the results shows, two stream configurations increase the accuracy of both architectures. Also, using attention modules for CNNs produce more global features. We find out that, compared to hybrid architecture, optical flow images give better results for 3D-CNNs. Additionally, pre-training the network on a large scale dataset increase the accuracy.

There are some possible future works in order to increase the recognition further. First of all, training the CNN-LSTM networks together (end-to-end training) could increase the accuracy. We didn't train the networks end to end, due to memory limitations. Although we have reached high accuracy values, it may reach higher when it is trained end to end. Also, before training the networks in our datasets, training in large datasets could help the networks to learn better features. Because of the high computation cost, networks are not trained on large dataset, like Kinetics, at first. Only for I3D, we have used Kinetics weights to initialize the network, which are published publicly. Lastly, we have observed that, using attention blocks produce better features. It might be useful to investigate different kind of attention blocks and their advantages.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *CoRR*, vol. abs/1406.2199, 2014.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[7] J. Walker, A. Gupta, and M. Hebert, "Dense optical flow prediction from a static image," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[8] D. Fortun, P. Bouthemy, and C. Kervrann, "Optical flow modeling and computation: A survey," *Computer Vision and Image Understanding*, vol. 134, p. 1–21, 2015.

[9] H. Ye, Z. Wu, R.-W. Zhao, X. Wang, Y.-G. Jiang, and X. Xue, "Evaluating two-stream cnn for video classification," *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval - ICMR 15*, 2015.

[10] S.-H. Tsang, ""review: Vggnet - 1st runner-up (image classification), winner (localization) in ilsvrc 2014."," tech. rep., Medium, Coinmonks, medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11., 19 Oct. 2020.

[11] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013. cite arxiv:1312.4400Comment: 10 pages, 4 figures, for iclr2014.

[12] A. Ng, "C4w2l06 inception network motivation," tech. rep., DeeplearningAI, 7 Nov 2017.

[13] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[14] A. Pandey, "Depth-wise convolution and depth-wise separable convolution, medium post," posted on Sep 9, 2018.

[15] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017.

[16] J. Hu, L. Shen, and G. Sun, "Squeeze and excitation networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[17] C. Olah, "Understanding lstm networks,github blog," posted on August 27 2015.

[18] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks, andrej karpathy blog," posted on May 21, 2015.

[19] A. Karpathy, "Recurrent neural networks tutorial, part 3 – backpropagation through time and vanishing gradients, denny britz," posted on OCTOBER 8, 2015.

[20] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[21] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] Y. Kong, Z. Tao, and Y. Fu, "Deep sequential context networks for action prediction," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] Y. Li, G. Lin, B. Zhuang, L. Liu, C. Shen, and A. V. D. Hengel, "Sequential person recognition in photo albums with a recurrent network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[24] M. Asadi-Aghbolaghi, A. Clapes, M. Bellantonio, H. J. Escalante, V. Ponce-Lopez, X. Baro, I. Guyon, S. Kasaei, and S. Escalera, "A survey on deep learning based approaches for action and gesture recognition in image sequences," *2017 12th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2017)*, 2017.

[25] Y. Kong and Y. Fu, "Action recognition and human interaction," *Human Activity Recognition and Prediction*, p. 23–48, 2015.

[26] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[27] A. Bobick and J. Davis, "The recognition of human movement using temporal templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 3, p. 257–267, 2001.

[28] D. Weinland, R. Ronfard, and E. Boyer, "Free viewpoint action recognition using motion history volumes," *Computer Vision and Image Understanding*, vol. 104, no. 2-3, p. 249–257, 2006.

[29] O. Koller, S. Zargaran, and H. Ney, "Re-sign: Re-aligned end-to-end sequence modelling with deep recurrent cnn-hmms," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[30] Q. Guo, F. Wang, J. Lei, D. Tu, and G. Li, "Convolutional feature learning and hybrid cnn-hmm for scene number recognition," *Neurocomputing*, vol. 184, p. 78–90, 2016.

[31] R. Cui, H. Liu, and C. Zhang, "Recurrent convolutional neural networks for continuous sign language recognition by staged optimization," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition.," in *ICML*, pp. 495–502, 2010.

[33] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers, "An improved algorithm for tv-l 1 optical flow," *Lecture Notes in Computer Science Statistical and Geometrical Approaches to Visual Motion Analysis*, p. 23–45, 2009.

[34] S. J. Zhao Yuxuan, Man Ka Lok, "Improved two-stream model for human action recognition," *EURASIP Journal on Image and Video Processing*, 2020/06/17.

[35] J. S. Pérez, E. Meinhardt-Llopis, and G. Facciolo, "Tv-l1 optical flow estimation," *Image Processing On Line*, vol. 3, p. 137–150, 2013.

[36] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l 1 optical flow," *Lecture Notes in Computer Science Pattern Recognition*, p. 214–223.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. cite arxiv:1409.1556.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[39] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," 2014.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[41] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, t. Kaiser, and I. Polosukhin, “Attention is all you need,” p. 5998–6008, 2017.

[43] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, p. 1735–1780, 1997.

[44] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning.,” *CoRR*, vol. abs/1602.07261, 2016.

[45] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman, “The kinetics human action video dataset,” *CoRR*, vol. abs/1705.06950, 2017.

CHAPTER 6

APPENDIX

Table 6.1: ChaLearn 40 Class Dataset (Part 1).

Class Index	**Class Group**	**Class Name**
0/C-021	ItalianGestures	Bellissima
1/C-027	ItalianGestures	NonMiFrega
2/C-030	ItalianGestures	VieniQua
3/C-049	GestunoDisaster	thunderstorm_orage
4/C-052	GestunoDisaster	earthquake_trembleme
5/C-065	GestunoLandscape	volcano_volcan
6/C-071	GestunoSmallAnimals	butterfly_papillon
7/C-072	GestunoSmallAnimals	cat_chat
8/C-086	GestunoTopography	harbour_port
9/C-177	HelicopterSignals	LiftOff
10/C-001	Mudra1	Ardhachandra
11/C-002	Mudra1	Ardhapataka
12/C-003	Mudra1	Chandrakala
13/C-006	Mudra1	Pataka
14/C-012	Mudra2	Chakram
15/C-013	Mudra2	Hamsapaksha
16/C-014	Mudra2	Mayura
17/C-015	Mudra2	Mrigashirsha
18/C-018	Mudra2	Tamrachuda
19/C-031	ChineseNumbers	ba

Table 6.2: ChaLearn 40 Class Dataset (Part 2).

Class Index	**Class Group**	**Class Name**
20/C-033	ChineseNumbers	liu
21/C-035	ChineseNumbers	shi
22/C-037	ChineseNumbers	wu
23/C-039	GestunoColors	colour_couleur
24/C-040	GestunoColors	black_noir
25/C-043	GestunoColors	yellow_jaune
26/C-046	GestunoColors	green_vert
27/C-050	GestunoDisaster	tide_maree
28/C-051	GestunoDisaster	dought_secheresse
29/C-054	GestunoDisaster	flood_inondation
30/C-056	GestunoDisaster	hurricane_ouragan
31/C-077	GestunoSmallAnimals	pigeon_pigeon
32/C-090	MusicNotes	do
33/C-092	MusicNotes	fa
34/C-119	CraneHandSignals	EverythingSlow
35/C-129	DivingSignals1	ComeHere
36/C-136	DivingSignals2	CannotOpenReserve
37/C-137	DivingSignals2	Cold
38/C-140	DivingSignals2	Meet
39/C-142	DivingSignals2	PressureBalancePb

Table 6.3: Number of training and test samples for each classes (Part 1).

Class Names	Training Samples	Test Samples
c001	151	33
c002	144	37
c003	150	46
c006	155	41
c012	157	27
c013	163	30
c014	157	45
c015	151	42
c018	297	94
c021	131	43
c027	144	34
c030	149	29
c031	160	48
c033	141	43
c035	156	35
c037	344	75
c039	115	23
c040	126	26
c043	119	21
c046	139	26

Table 6.4: Number of training and test samples for each classes (Part 2).

Class Names	Training Samples	Test Samples
c049	158	45
c050	155	52
c051	156	45
c052	163	41
c054	144	41
c056	157	42
c065	68	20
c071	125	28
c072	117	22
c077	112	35
c086	70	24
c090	210	47
c092	318	72
c119	96	14
c129	111	22
c136	104	30
c137	99	26
c140	104	28
c142	105	24
c177	151	37